EL PROBLEMA DE ASIGNACIÓN GENERALIZADO DINÁMICO: MODELACIÓN Y ESTRUCTURA.

Por

Ronald David Suárez Díaz

Ingeniero Industrial
Universidad del Norte
Barranquilla, Colombia. 2012

Remitido al Departamento de Ingeniería Industrial para el cumplimiento de los requerimientos para el grado de

Máster en Ingeniería Industrial

UNIVERSIDAD DEL NORTE

Firma del Autor ……

Maestría Ingeniería Industrial,
División de ingeniería Industrial
Septiembre, 2012

Aceptada por ……

Carlos Paternina Arboledar, Ph.D.
División Escuela de Negocios

EL PROBLEMA DE ASIGNACIÓN GENERALIZADO DINÁMICO: MODELACIÓN Y ESTRUCTURA.

RONALD DAVID SUÁREZ DÍAZ

Director: Carlos Paternina Arboleda

Ingeniero Industrial, Ph.D.

Codirector: Miguel Rojas Santiago

Ingeniero Industrial, Ph.D.

BARRANQUILLA

UNIVERSIDAD DEL NORTE
DIVISIÓN DE POSTGRADOS E INVESTIGACIONES EN INGENIERIA
MAESTRIA EN INGENIERIA INDUSTRIAL
2012

TABLA DE CONTENIDOS

LISTA DE ECUACIONES

LISTA DE GRÁFICOS

LISTA DE TABLAS

LISTA DE EJEMPLOS

INTRODUCCIÓN

Los Problemas de Asignación (de ahora en adelante AP, por sus siglas en inglés) han jugado un papel muy importante dentro de la optimización de problemas de la vida real, ya que son múltiples las situaciones que pueden modelarse de este modo.

Son muchas las variantes del AP, dependiendo de las características del problema bajo consideración; sin embargo, al pertenecer estas variantes a la clase NP, resulta difícil abordarlos como un problema de optimización (o de valor óptimo), ya que es improbable encontrar la mejor solución en tiempo polinomial; así, en muchas ocasiones, basta con encontrar una respuesta que sea lo "suficientemente cercana al óptimo", en un tiempo de cómputo razonable.

En este trabajo se define una nueva variante del AP, denominada *Problema de Asignación Generalizado Dinámico* (de ahora en adelante DGAP). Se deducen algunas propiedades matemáticas del mismo, que luego son usadas en la elaboración de un algoritmo genético empleado para su solución.

1 GENERALIDADES DEL PROYECTO

1.1 ANTECEDENTES

Muchos problemas de la vida real pueden ser solucionados mediante programación matemática, entre los cuales se puede citar los de asignación. Cada uno de estos tiene una estructura matemática bien definida, que permite establecer algoritmos eficientes para su solución. Sin embargo, cuando no se logra asociar un problema de programación matemática real con algún problema teórico conocido, su solución se da por medio de los métodos generales más usados (Método Simplex, Métodos de Punto Interior; Branch and Bound, Branch and Cut, entre otros), los cuales pueden tomar mucho tiempo de cómputo. Lo anterior se hizo palpable durante la ejecución de un proyecto en el grupo Mundial Logistics Service (2011-2012), en donde la optimización de la red de distribución trajo consigo un modelo matemático que resultaba en sí mismo muy difícil de resolver. Pese a que se trataba de un problema de asignación (Determinar que clientes debían ser atendidos desde cada centro de distribución) las restricciones impuestas (nivel de servicio y demanda cambiante en el tiempo) hacían imposible asociarlo con alguno de los problemas de asignación conocidos.

De lo anterior, es claro que deben ser muchas las situaciones de la vida real donde se presenta el inconveniente ya mencionado, el cuál evidentemente puede ocasionar una pérdida de tiempo considerable durante la solución del problema bajo estudio.

1.2 FORMULACIÓN DEL PROBLEMA

El Problema de Asignación pertenece a la rama de la Optimización Combinatoria (Schrijver, 1990). Éste consiste, de manera muy general, en asignar un conjunto de tareas a un conjunto de agentes, de modo que cada tarea solo puede ser atendida por un único agente y el costo total de asignación sea mínimo.

Muchas han sido las variantes que han surgido del AP, dependiendo de las restricciones y características del problema que se esté estudiando. Como ya se mencionó con anterioridad, resulta necesario indagar sobre la existencia de alguna variante donde se contemplen número de requerimientos no atendidos (nivel de servicio), requerimientos de tareas variantes en el tiempo y capacidad variante y no instantánea en los agentes (se entiende por no instantáneo el hecho de que toma cierto tiempo disponer de los recursos de los agentes).

La variante en donde es permitido que un agente procese más de una tarea, y donde éstos tienen capacidad limitada, se conoce como *Problema de Asignación Generalizado.* El GAP fue introducido por primera vez por Ross & Soland (1975), inspirados en problemas de la vida real tales como asignación de trabajos a redes computacionales (Balachandran, 1976).

De las variantes del GAP más cercanas a capacidad variable en los agentes con disponibilidad no instantánea, se tiene en la literatura el Problema de Asignación Generalizado Multi Nivel (MLGAP, por sus siglas en inglés) introducido por Laguna et al. (1995), el Problema de Asignación Dinámico Multi Recurso, propuesto por Shtub y Kogan (1998), el Problema de Multi Asignación Generalizado de Park et al. (1998), entre otras.

Sin embargo, de la revisión de la literatura, no se encontró ninguna variante del GAP que satisfaga las siguientes condiciones:

- Capacidad de agentes dependiente del tiempo
- Capacidad de agentes dependiente de un agente principal
- Tasas de consumo de recursos variables en el tiempo
- Cantidad mínima de requerimientos de los agentes que debe ser atendida.

Por tanto, en este proyecto se hace la definición matemática de una nueva variante del AP, más específicamente, una variante del GAP, la cual tiene en cuenta las características anteriormente señaladas. A esta variante se le denomina *Problema de asignación Generalizado Dinámico* (DGAP). La elección del nombre se debe a lo siguiente:

- ✓ Asignación: El problemas consiste en asignar un conjunto de tareas a un conjunto de agentes.
- ✓ Generalizado: Un Agente puede atender más de una tarea a la vez.
- ✓ Dinámico: Los requerimientos de las tareas cambian a lo largo del tiempo.

Sin embargo, este no es el fin último de este proyecto. Una vez planteado el modelo matemático del nuevo problema, resulta interesante analizar sus propiedades matemáticas, con el ánimo de encontrar un algoritmo (o un conjunto de los mismos) que permita solucionarlo de manera eficiente. Para el caso en estudio, se opta por la utilización de Algoritmos Genéticos, ya que han probado ser muy potentes en la solución de múltiples problemas de optimización combinatoria de gran tamaño, además de que el DGAP resulta ser un poco extenso en estructura, por lo que abordarlo con métodos exactos en su totalidad resultaría ser muy dispendioso. La idea básica es utilizar las propiedades matemáticas para mejorar el rendimiento del algoritmo, mediante el principio de manipulación genética. A esta modificación se le denominará *Algoritmo Genético Potenciado.*

Basados en todo lo anterior, este proyecto busca responder la pregunta: ¿Qué tan eficiente en calidad de respuesta y en tiempo de ejecución computacional resulta

ser la aplicación de un Algoritmo Genético Potenciado al DGAP para el caso mono-objetivo?

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Proponer un Modelo Matemático para el DGAP, deduciendo sus propiedades matemáticas más importantes, permitiendo establecer condiciones para facilitar la solución de cualquier instancia del problema.

1.3.2 OBJETIVOS ESPECÍFICOS

- Desarrollar un algoritmo genético para resolver DGAP
- Establecer una estructura teórica especial de selección de los operadores de selección, cruzamiento y mutación del algoritmo.
- Calcular el orden de complejidad computacional del algoritmo.
- Determinar la complejidad computacional del DGAP.
- Determinar la convergencia experimental del algoritmo.
- Implementar el algoritmo propuesto en un caso de estudio de la vida real.

1.4 JUSTIFICACIÓN

El problema de asignación y sus variantes son de amplio interés en el campo investigativo, dado su alto número de aplicaciones, entre las que se puede resaltar asignación de trabajos a máquinas, asignación de personal a tareas, enrutamiento de vehículos, procesamiento de señales, asignación de contratos a licitadores, asignación de profesores a clases, entre muchas otras, por lo que el desarrollo de herramientas que solucionen de manera eficiente y con un nivel de calidad aceptable dichos problemas cobra cada vez mayor importancia.

Como ya se mencionó anteriormente, el DGAP no se encuentra definido en la literatura. Entre las aplicaciones más relevantes que pueden citarse del DGAP se tienen:

- Asignación de clientes a centros de distribución, donde la capacidad de estos últimos está sujeta al nivel de producción de una o varias plantas de producción. Existe un tiempo de transporte desde cada origen hasta cada destino, y cada centro ofrece a cada cliente una promesa de entrega.
- Asignación de puntos geográficos (Ciudades, municipios) a subestaciones eléctricas, donde la capacidad de cada subestación en un mes, por ejemplo, depende de la energía que le suministra una planta principal. De hecho, la subestación principal también podría atender regiones directamente.
- Asignación de trabajos a máquinas en paralelo (o estaciones en paralelo), donde, para poder funcionar, cada máquina requiere de suministro de materias primas de un conjunto de estaciones o almacenes. Dichos

almacenes tienen capacidad limitada. Las máquinas pueden tener ritmos de trabajo iguales (P_m), proporcionales (Q_m) o no relacionados (R_m).

Así, dado el alto nivel de aplicación del DGAP en problemas de la vida real, surge la necesidad de la definición de este nuevo problema de asignación, así como el deber paralelo de proponer un algoritmo que resulte eficiente tanto en calidad de respuesta como en tiempo de cómputo, para la solución del mismo. Acoplando todas las consideraciones anteriores, surge el problema de investigación aquí tratado.

Una objeción al planteamiento del problema de investigación, de hecho, la más fundamentada, resulta ser: ¿Por qué hacer énfasis en este problema, si puede modelarse matemáticamente y resolverse con las técnicas de programación entera mixta ya existentes? A esto se respondería, de forma clara, con el siguiente argumento: "*Sería como tratar de resolver un problema de mecánica clásica aplicando teoría de la relatividad. Claramente, la segunda teoría reduce a la primera; sin embargo, la mecánica clásica tiene su propia estructura especial (Leyes de Newton), la cual permite resolver el problema de forma más simple*". Así, al aislar el GAP con la estructura de interés sentada en el problema de investigación, sus características particulares permitirán establecer condiciones de solución más rápidas, con altos niveles de eficiencia. Como ejemplo puede citarse el problema de la mochila, el cual, de inicio, no es resuelto como un modelo de programación entero mixto, ya que, en lugar de eso, se parte de ciertas cotas características propias de la estructura del problema, como la expuesta por Dantzig-Wolfe (1960).

1.5 ALCANCES Y LIMITACIONES

Este proyecto no va más allá de la propuesta del modelo matemático del DGAP, el análisis de su estructura matemática y la proposición de un algoritmo genético modificado para su solución en el caso de un solo objetivo. No está dentro del alcance del proyecto trabajar el DGAP multiobjetivo.

Dentro las limitaciones del proyecto, se encuentra la solución de instancias grandes del problema estudiado de manera exacta (Para la comparación con el algoritmo propuesto) ya que la complejidad del problema crece exponencialmente con su tamaño, de manera que solucionar instancias demasiado grandes requeriría un gasto considerable de tiempo. Por tal razón, se trabajó con instancias de máximo 500 tareas, 20 agentes y 100 instantes de tiempo, las cuales son, en esencia, de gran tamaño.

1.6 METODOLOGÍA

En primer lugar, la investigación que aquí se lleva a cabo es del tipo descriptiva-deductiva, pues, se inicia describiendo matemáticamente el DGAP, hasta deducir sus características matemáticas y plantear el algoritmo genético adecuado para su solución.

La metodología a seguir durante la investigación es la siguiente:

1) Construcción del marco teórico de la investigación.

2) Indagación sobre el estado del arte de los Problemas de Asignación, sus variantes y metodologías de solución propuestas para el caso mono-objetivo.

3) Planteamiento del Modelo Matemático del DGAP, y análisis posterior de su estructura matemática.

4) Formulación del EGA, diseño de su metodología, determinación de su complejidad computacional y análisis de convergencia experimental.

5) Una vez validado (análisis de convergencia), se implementará el EGA en un problema del tipo DGAP de la vida real. La empresa donde se implementará la aplicación es Compañía Global de Pinturas (CGP), la cual es una filial de Pintuco en Colombia.

Finalizados todos los pasos de la metodología, se estará en capacidad de analizar el cumplimiento de los objetivos planteados en el problema de investigación.

2 MARCO DE REFERENCIA

2.1 MARCO TEORICO

A continuación se definen los conceptos y teorías básicas usadas en este proyecto. Comenzando con la definición de Algoritmo, y continuando con las bases teóricas del problema tratado aquí, como lo son Teoría de la Complejidad , el Problema de Asignación (AP), Problema de Asignación Generalizado (GAP), y Algoritmos Genéticos.

2.1.1 ALGORITMO

Se define como algoritmo un conjunto de pasos finitos, claros, precisos y no ambiguos usados para la resolución de un problema. Todo algoritmo requiere de una entrada, que consiste de los datos del problema que se desea resolver, y de una salida, que muestra la solución que arroja el algoritmo para el problema. Dicho esquema se ilustra en el Gráfico 1.

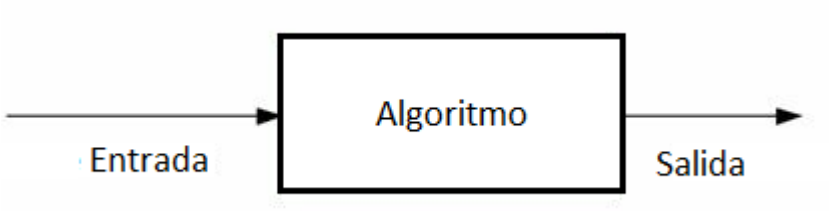

Gráfico 1: Flujo de información en un algoritmo.

Los algoritmos suelen estar relacionados con la computación, ya que estos son escritos en lenguaje máquina para resolver diversos problemas de gran tamaño que no pueden ser tratados manualmente. Sin embargo, la estructura básica parte siempre de la mente humana. Dado que el lenguaje humano difiere del lenguaje máquina, en inicio, el algoritmo debe ser escrito de modo que un ser humano pueda entender su proceder. El lenguaje en el que un algoritmo es escrito para el entendimiento de las personas se denomina pseudocódigo.

Existen diversas clasificaciones para los tipos de algoritmos. En general, estos pueden clasificarse de dos modos: Según el fin y según el tipo. Existen dos sub clasificaciones para ambas categorías. Sin embargo, muchos algoritmos no tienen una clasificación bien definida, y puede que resulten en apariencia pertenecer a diversas clasificaciones al mismo tiempo.

2.1.1.1 CLASIFICACIÓN DE ALGORITMOS SEGÚN EL FIN

Según el fin, los algoritmos pueden ser clasificados como de búsqueda o de ordenamiento. Los primeros recuperan información de la entrada sin alterarla, mientras que los segundos alteran de alguna manera los datos de entrada para poder darles cierto orden.

- Ejemplo 1: Considere el problema de buscar el máximo valor en un vector v de n posiciones. El algoritmo necesario para ello sería el siguiente (Se trabaja con algo de notación en C):

$Paso\ 1: Hacer\ máximo = -M; i = 0.$

$Paso\ 2: Si\ máximo < v[i]\ hacer\ \ máximo = v[i]\ e\ i = i + 1; sino\ hacer\ i = i + 1.$

$Si\ i = n\ terminar\ y\ retornar\ máximo, sino\ repetir\ el\ paso\ 2.$

Como puede verse, se trata de un algoritmo de búsqueda, ya que se recupera el valor máximo del arreglo sin necesidad de alterar la configuración inicial del mismo.

- Ejemplo 2: Se tiene un vector v de n posiciones y se requiere ordenar sus elementos de menor a mayor (o viceversa). Un algoritmo muy utilizado para este fin se denomina *ordenamiento burbuja*, y su pseudo código se describe a continuación:

$Paso\ 1: Hacer\ i = 0, k = 0.$
$Paso\ 2: Si\ v[i] > v[i+1] intercambiar\ las\ posiciones\ y\ hacer\ k = 1. Hacer\ i = i + 1$
$Paso\ 3: Si\ i = n\ y\ k = 1\ volver\ al\ paso\ 1;\ Si\ i = n\ y\ k = 0, entonces\ terminar.$

Según lo anterior, el ordenamiento burbuja es un algoritmo de ordenamiento, pues para poder ordenar los datos era necesario alterar su estado inicial.

2.1.1.2 CLASIFICACIÓN DE ALGORITMOS SEGÚN EL TIPO

Según el tipo, los algoritmos pueden ser determinísticos o estocásticos[1]. Los primeros tienen pasos perfectamente definidos, y si se ejecutan varias veces para la misma entrada el resultado será siempre el mismo. Los segundos tienen pasos en función de valores pseudoaleatorios, y si se ejecutan varias veces para una misma entrada el resultado no siempre tiene porque ser igual.

- Ejemplo 3: Considere el algoritmo de búsqueda del máximo de un vector tratado en el Ejemplo 1. Se trata claramente de un algoritmo determinístico, ya que, sus pasos no dependen de valores pseudoaleatorios, y para una misma entrada, el resultado del máximo siempre será el mismo, independientemente del número de veces que se ejecute el algoritmo.

[1] Tomado de www.wikipedia.com

- Ejemplo 4: Considere un algoritmo que pretende estimar el valor esperado de una variable aleatoria uniformemente distribuida entre 0 y 1 a partir de la generación de 5 datos. Su pseudocódigo se ilustra a continuación:

$Paso\ 1: Hacer\ i = 0, Suma = 0.$

$Paso\ 2. Si\ i = 4, hacer\ Media = \frac{Suma}{5}\ y\ terminar; sino\ hacer\ Suma$

$= Suma + Rand(0{,}1), i = i + 1.$

Se corre dos veces el algoritmo, generando los números pseudoaleatorios en una calculadora. Los resultados se ilustran en el Tabla 1:

	Primera Corrida	Segunda Corrida
Rand(0)	0,248466396	0,50670605
Rand(1)	0,324281121	0,040705616
Rand(2)	0,93752785	0,241568693
Rand(3)	0,873458891	0,038708012
Rand(4)	0,414838549	0,316652245
Suma	2,798572807	1,144340617
Promedio	0,559714561	0,228868123

Tabla 1: Ejemplo de algoritmo estocástico

Como puede apreciarse, los resultados dependen en cada paso del valor pseudoaleatorio generado, por lo que el promedio obtenido es distinto en ambas corridas. Por tanto, se trata de un algoritmo estocástico.

2.1.2 COMPLEJIDAD COMPUTACIONAL

En el análisis de algoritmos, hay dos factores que resultan de vital interés.; estos son la eficiencia y la calidad. El primero hace alusión al tiempo tomado por el algoritmo para resolver el problema, y el segundo es una medida de que tan cerca se encuentra la respuesta obtenida de la respuesta óptima. A continuación se detalla teóricamente cada uno de estos factores:

2.1.2.1 EFICIENCIA

En muchos problemas abordados mediante algoritmia, el tiempo de solución juega un papel muy importante, por lo que resulta útil tener una medida del tiempo que un algoritmo tarda en resolver un determinado problema. Sin embargo, dado que la solución de un mismo problema, por el mismo algoritmo, puede variar entre una computadora y otra, se necesita otra medida; esta medida resulta ser el número de pasos, ya que, independiente de la computadora donde se ejecute el algoritmo, el número de pasos necesarios para resolver el problema siempre será el mismo. El número de pasos empleado por el algoritmo se representara mediante una función del tamaño del problema, a saber, $f(n)$. Se introducen ahora los conceptos fundamentales en el acotamiento del número de pasos:

2.1.2.1.1 ORDEN DE COMPLEJIDAD O

Sean $f(n)$ y $g(n)$ funciones definidas sobre los números naturales. Se dice que $f(n)$ es de orden máximo $g(n)$, o simplemente $f(n) = O\big(g(n)\big)$, si existen constantes $c > 0$ y $n_0 \in N$, tales que:

$$f(n) \leq cg(n);\ \forall\, n \geq n_0 \quad [1]$$

- Ejemplo 5: Sea $f(n) = n^2 + 2n + 3$. Probar que $f(n) = O(n^2)$.

Solución:

$$1 \leq n;\ \forall\, n \in N.$$

$$n \leq n^2;\ \forall\, n \in N.$$

Usando el último argumento:

$$n^2 + 2n + 3 \leq n^2 + 2n^2 + 3n^2 = 6n^2;\ \forall\, n \in N$$

Dado que $c = 6$, $n_0 = 1$ y $g(n) = n^2$, se concluye que $f(n) = O(n^2)$.

Lo anterior indica que un algoritmo que tenga como número de pasos a $f(n)$, tendrá, como máximo, una cantidad de pasos equivalente a $cg(n)$, a medida que n

tiende a infinito. En otras palabras, $cg(n)$ acota el número de pasos de $f(n)$. Para el ejemplo 5, se tendría una representación gráfica como la que sigue:

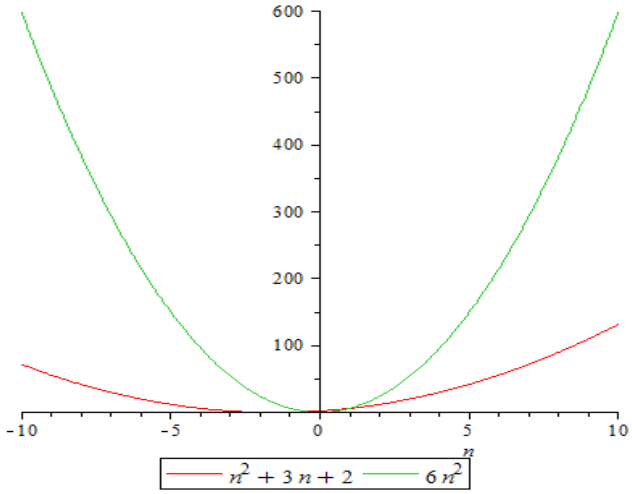

Gráfico 2: Representación del Ejemplo 5.

- Ejemplo 6: Sea $f(n) = n + \frac{n}{2} + \frac{n}{4} + \cdots + 1$, siendo n una potencia de 2. Demuestre que $f(n) = O\big(nLn(n)\big)$.

Solución: Para cualquier constante $k \geq 1$ se tiene que:

$$\frac{n}{k} \leq n; \quad \forall\, n \in N$$

Usando el anterior argumento:

$$n + \frac{n}{2} + \frac{n}{4} + \cdots + 1 \leq n + n + \cdots + n$$

Sea r el número de términos en la suma. Se tiene que:

$$\frac{n}{2^{r-1}} = 1$$

$$r - 1 = Ln(n)/Ln(2)$$

$$r = \frac{Ln(n)}{Ln(2)} + 1$$

Así:

$$n + \frac{n}{2} + \frac{n}{4} + \cdots + 1 \leq n\left(\frac{Ln(n)}{Ln(2)} + 1\right) \leq 2nLn(n); \quad \forall n \geq 3$$

Luego, dado que $c = 2$, $n_0 = 3$ y $g(n) = 2nLn(n)$, se concluye que $f(n) = O\big(nLn(n)\big)$:

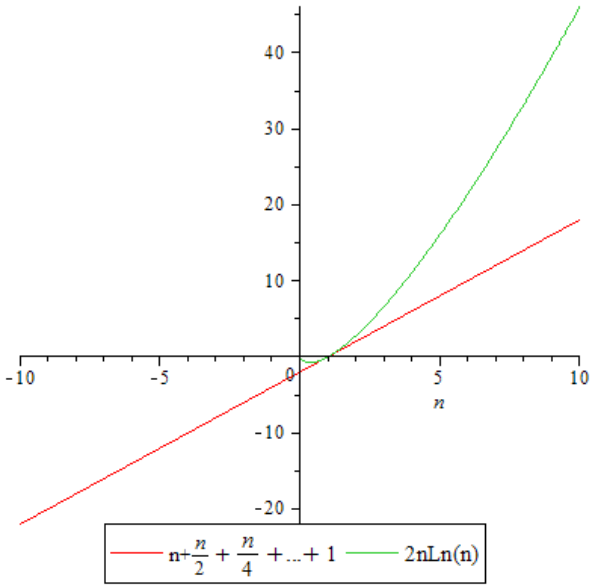

Gráfico 3: Representación Ejemplo 6.

A continuación se listan de menor a mayor los órdenes máximos de complejidad más relevantes (Cualquier función logarítmica, sin importar la base, se representará por $Log(n)$):

$O(C)$: *Constante*

$O\big(LogLog(n)\big)$: *Log Log*

$O\big(Log(n)\big)$: *Log*

$O(n^c), 0 < c < 1$: *Sublineal*

$O(n)$: *Lineal*

$O\big(nLog(n)\big)$: *n Log n*

$O(n^2)$: *Cuadrático*

$O(n^3)$: *Cúbico*

$O(n^k), k \geq 1$: *Polinomial*

$O(c^n), c > 1$: *Exponencial*

$O(n!)$: *Factorial*

Gráfico 4: Ordenes máximos de Complejidad

Un algoritmo se considera *eficiente* si su orden máximo de complejidad es, a lo sumo, polinomial (por ejemplo $O(n^4)$). Sin embargo, para valores de k muy grandes (como $k = 30$) un algoritmo ya se considera medianamente eficiente.

2.1.2.1.2 ORDEN DE COMPLEJIDAD Ω

Sean $f(n)$ y $g(n)$ funciones definidas sobre los números naturales. Se dice que $f(n)$ es de orden mínimo $g(n)$, o simplemente $f(n) = \Omega\big(g(n)\big)$, si existen constantes $c > 0$ y $n_0 \in N$, tales que:

$$f(n) \geq cg(n);\ \forall\, n \geq n_0 \quad [2]$$

La anterior indica una cota inferior para el número de pasos que un algoritmo ejecuta.

- Ejemplo 7: Sea $f(n)$ como se definió en el Ejemplo 5. Demuestre que $f(n) = \Omega(n^2)$.

Solución: Mediante Operaciones algebraicas simples se tiene:

$$n^2 + 2n + 3 = (n^2 + 2n + 1) + 2 = (n + 1)^2 + 2$$
$$\geq (n + 1)^2 \geq n^2$$

La prueba ha sido completada. Gráficamente:

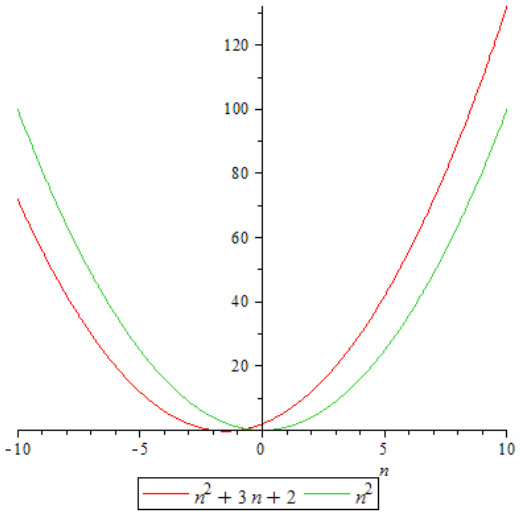

Gráfico 5: Representación Ejemplo 7.

Los órdenes de complejidad mínimos son idénticos a los máximos, variando el símbolo O por Ω.

2.1.2.1.3 ORDEN DE COMPLEJIDAD Θ

Sean $f(n)$ y $g(n)$ funciones definidas sobre los números naturales. Se dice que $f(n)$ es de orden $g(n)$, o simplemente $f(n) = \Theta\big(g(n)\big)$, si $f(n) = O(g(n))$ y $f(n) = \Omega\big(g(n)\big)$.

Ejemplo 8: Sea $f(n)$ como en los Ejemplos 5 y 7. Claramente $f(n) = \Theta\big(g(n)\big)$, ya que se ha probado que $f(n) = O(g(n))$ y que $f(n) = \Omega\big(g(n)\big)$:

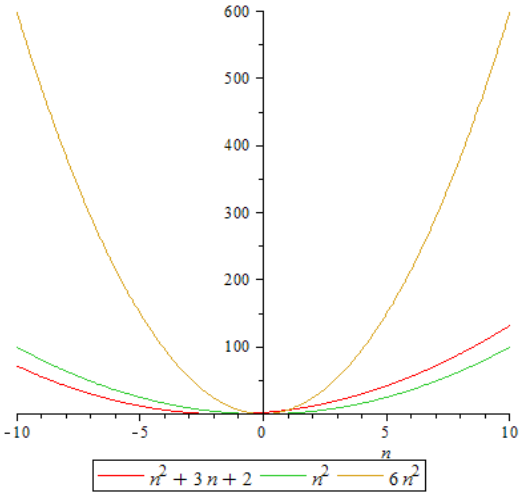

Gráfico 6: Representación Ejemplo 8.

$\Theta(C)$: *Constante*

$\Theta(LogLog(n))$: *Log Log*

$\Theta(Log(n))$: *Log*

$\Theta(n^c), 0 < c < 1$: *Sublineal*

$\Theta(n)$: *Lineal*

$\Theta(nLog(n))$: *n Log n*

$\Theta(n^2)$: *Cuadrático*

$\Theta(n^3)$: *Cúbico*

$\Theta(n^k), k \geq 1$: *Polinomial*

$\Theta(c^n), c > 1$: *Exponencial*

$\Theta(n!)$: *Factorial*

Gráfico 7: Órdenes de Complejidad

El orden de complejidad más usado en el análisis de algoritmos es el orden máximo, ya que este determina el comportamiento del algoritmo en el peor de los casos.

2.1.2.2 CALIDAD

El análisis de la eficiencia algorítmica no es del todo suficiente. También es de gran importancia determinar que tan buenas son las respuestas dadas por el algoritmo a diversas instancias de un problema. Tal como se analiza el peor caso del número de pasos, también se analiza el peor caso en respuesta. La medida para el peor caso en lo que concierne a la calidad de un algoritmo, se denomina *Garantía de Desempeño,* y se define a continuación:

2.1.2.2.1 GARANTÍA DE DESEMPEÑO

Sea f_{opt} la respuesta óptima a un problema de minimización (o maximización) para alguna instancia del mismo, y sea f_{alg} la respuesta dada por el algoritmo. Se dice que el algoritmo tiene garantía de desempeño dada por un real positivo k, si se cumple que:

$$k \geq \frac{f_{alg}}{f_{opt}} \geq 1; k \geq 1 \ (Caso\ Minimización) \qquad [3]$$

$$k \leq \frac{f_{alg}}{f_{opt}} \leq 1; 0 < k \leq 1 \ (Caso\ Maximización) \quad [4]$$

Así, la garantía de desempeño es una medida de la peor respuesta que puede obtener un algoritmo para cualquier instancia del problema que se esté estudiando.

2.1.2.3 MÁQUINA DE TURING

Una Máquina de Turing (De ahora en adelante MT) es una máquina ideal, que transforma una entrada (la cual se mueve a lo largo de una cinta) en una salida después de un número de pasos; en otras palabras, se trata en sí de un algoritmo.

Fue propuesta por Alan Turing en 1936, y no se trata de un dispositivo real, sino de una máquina hipotética que representa una computadora. La finalidad de la máquina era resolver el problema de la decidibilidad matemática propuesto por David Hilbert .

Una representación simple para una MT es la siguiente:

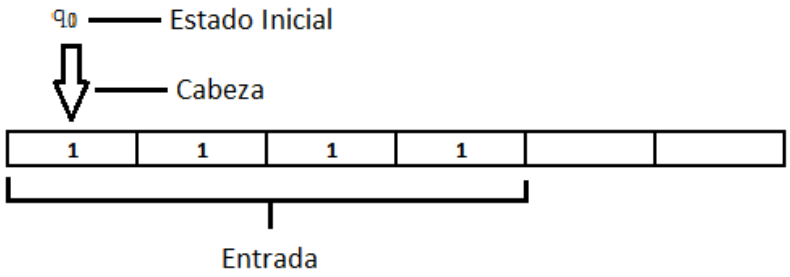

Gráfico 8: Ejemplo de Máquina de Turing

La cabeza se ubica siempre en la casilla más hacia la izquierda de la cinta, en donde se encuentra el estado inicial. Después de la entrada, la cinta consta hacia la derecha de espacios en blanco. La cinta puede ser infinita hacia ambos lados, dependiendo de la situación.

Una MT con una sola cinta puede ser definida a través de una 7-Tupla $T = (Q, \Sigma, \Gamma, \delta, q_0, q_y, q_n)$, donde:

- $Q: Conjunto\ finito\ de\ estados$
- $\Sigma: Es\ el\ alfabeto\ de\ entrada$
- $\Gamma: Alfabeto\ de\ la\ cinta$
- $\delta: QxI\Gamma \rightarrow QxI\Gamma x\{I, D\} Función\ de\ transición$
- $q_0: Estado\ inicial$
- $q_y: Estado\ aceptador$

- q_N: $Estado\ de\ Rechazo$

En la función de transición, la pareja {I,D} indica movimiento a la izquierda o derecha en la cinta, respectivamente.

- Ejemplo 9: Se tiene un alfabeto de cinta $\Gamma = \{0,1\}$. La entrada también está escrita en términos de cierta cantidad de unos ($\Sigma = \{1\} \subset \Gamma$). En la cinta, cualquier celda vacía representa un cero. La MT hará una réplica exacta de la entrada después del primer cero que encuentre. Considere la entrada descrita en el Gráfico 8, en donde $q_0 = E_0$. Pueden definirse seis posibles estados de la máquina: $Q = \{E_0, E_1, E_2, E_3, E_4, E_5\}$.

Una vez definido el conjunto de estados de la máquina, se debe establecer la función de transición δ. Esta se define a continuación de manera tabular, como se ilustra en la Tabla 2:

Estado Actual	Carácter leído	Símbolo escrito	Nuevo estado	Movimiento
E_0	1	1	E_0	D
E_0	0	0	E_1	D
E_1	0	1	E_2	D
E_2	0	1	E_3	D
E_3	0	1	E_4	D
E_4	0	1	E_5	D

Tabla 2: Conjunto de estados y acciones de la MT del Ejemplo 9

Puede interpretarse la función de transición como sigue: Si estado en el estado inicial, se lee un 1, aún no se puede iniciar la replicación; si se encuentra un cero, se puede iniciar con la replicación del primer uno leído (Pasar al estado 1). Así se continúa hasta llegar al estado 5, en donde la máquina no tendría ninguna instrucción que ejecutar por lo que se detendría. Esto se representa en el Tabla 3, donde el color azul indica la posición del cabezal de la máquina.

E0	E0	E0	E0	E0	E1	E2	E3	E4	E5
1	1	1	1						
1	1	1	1						
1	1	1	1						
1	1	1	1						
1	1	1	1	0					
1	1	1	1	0	1				
1	1	1	1	0	1	1			
1	1	1	1	0	1	1	1		
1	1	1	1	0	1	1	1	1	
1	1	1	1	0	1	1	1	1	

Tabla 3: Ilustración Ejemplo 9

Más adelante, se mostrará la importancia que tiene la MT en el análisis de complejidad de problemas. En general, existen dos tipos de problemas: Los de decisión y los de optimización. Los primeros hacen la siguiente pregunta: *" ¿ Existe una solución S al problema tal que $f(S) \leq f_0$ (ó $f(S) \geq f_0$ en el caso de maximización) ?"* Evidentemente, la respuesta solo puede ser sí o no. Los problemas de optimización se describen como: "Encontrar una solución S al problema *tal que $f(S)$ sea mínimo(ó máximo) "*. Como puede verse, si se resuelve un problema de optimización, también se resuelve el problema de decisión, pero esto no se cumple en sentido inverso. De ahora en adelante se asumirá que las MT son empleadas para resolver el primer tipo de problemas.

Existen dos tipos de Máquinas de Turing. La Determinista (MTD) y la no Determinista (MTND). Para la primera, Cada pareja Estado-Símbolo tiene como una posibilidad de ejecución, mientras que en la segunda, cada pareja Estado-Símbolo tiene como mínimo dos posibilidades de ejecución; sin embargo, la máquina puede ejecutar en paralelo dichas posibilidades. Dicho de otro modo, una MTND evalúa todos los posibles movimientos de forma simultánea en anchura:

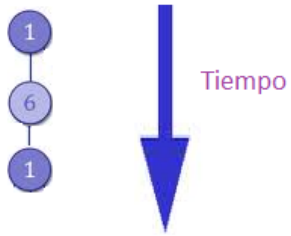

Gráfico 9: Máquina de Turing Determinista

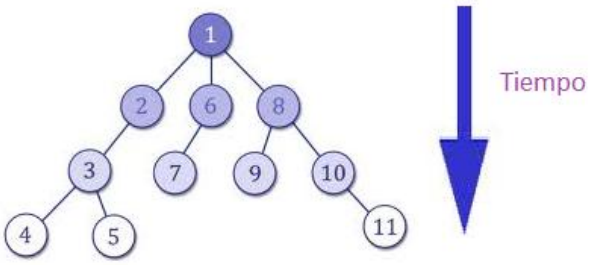

Gráfico 10: Máquina de Turing No Determinista

Entonces, lo que a una MTND le tomaría un tiempo $O(n^k)$, a una MTD le tomaría un tiempo $O(2^n)$. De cualquier modo, una MTD puede simular a la perfección una MTND, aunque con una mayor complejidad computacional. Los detalles de este hecho están fuera del alcance de este trabajo.

Para la verificación de la clausula de decisión, la MTND "adivina" un camino y verifica si este cumple la condición (Un camino de cada tallo a cada raíz, adivinando el más opcionado por tallo).

- Ejemplo 10: Considere el alfabeto $\Gamma = \{0,1\}$. Se tiene una entrada con alfabeto $\Sigma = \{0,1\}^n$ y se quiere verificar lo siguiente: ¿Existe algún elemento $S \in \{0,1\}^n$ tal que S satisfaga una condición P? Claramente, se tiene que existen 2^n posibles soluciones S. Como una MTD evalúa una posibilidad a la vez (ver gráfico 9) su orden máximo de complejidad para la verificación de la condición es $O(2^n)$. Sin embargo, una MTND puede en cada paso i avanzar al nodo número i de cada tallo (0 ó 1), generando al tiempo las 2^{i-1} ramas correspondientes a cada tallo. Entonces, generar todas las ramas requiere de n pasos, y como para cada tallo se adivina una solución (la más opcionada), la MTND tiene una complejidad $O(n)$.

2.1.2.4 PROBLEMAS *P* VS PROBLEMAS *NP*

En optimización combinatoria, además de estudiar las características de los algoritmos, también es indispensable estudiar la estructura o complejidad del problema que se está analizando.
Los problemas que pueden ser resueltos mediante un algoritmo polinomial se denominan tratables; en caso contrario, se denominan intratables.

- Ejemplo 11: Remítase al Ejemplo 1. En cada paso se hacen 2 comparaciones y máximo 2 asignaciones, un total de n veces, por lo que $f(n) = 4n = O(n)$. Luego dicho problema es tratable.

- Ejemplo 12: Considere el problema de encontrar todas las permutaciones posibles de un conjunto inicial de tamaño n. El algoritmo necesario implicaría enumerar todas las posibles $n!$ permutaciones posibles, razón por la cual este problema es intratable, debido a que $n!$ crece desmesuradamente a medida que n se incrementa ($n!$ no es una función polinomial, sino factorial).

Muchos problemas en la vida real, no pueden ser clasificados como tratables o intratables, por lo que surge una nueva teoría de clasificación. La de problemas ***P*** y ***NP.***

2.1.2.4.1 PROBLEMAS *P*

P es el conjunto de todos los problemas de decisión que pueden ser resueltos por un algoritmo polinómico (problemas tratables), o, dicho de otro modo, pueden ser resueltos por una MTD en tiempo polinomial.

- Ejemplo 13: Remítase al problema del ejemplo 11. Entonces, el problema de buscar el valor máximo en un vector pertenece a la familia ***P.***

2.1.2.4.2 PROBLEMAS *NP*

NP es el conjunto de todos los problemas de decisión que pueden ser resueltos en una MTND en tiempo polinómico (De allí la sigla NP: Nondeterministic Polynomial).

- Ejemplo 14: Un problema muy famoso en optimización combinatoria es el Problema del Agente Viajero (TSP, por sus siglas en Inglés), el cual, en su versión tipo decisión, se enuncia del siguiente modo: Se tiene un conjunto de n ciudades, con distancia entre par de ciudades $d_{ij}, i \neq j$. ¿Existe un Ciclo Hamiltoniano completo cuyo recorrido sea máximo D ?. A ciencia cierta, no se podría saber si este problema está en ***P***, pero si es cierto que es ***NP***, ya que una MTND tendría una complejidad $O(n^2)$ para resolverlo. De lo que se hablo respecto a las Máquinas de Turing, y de las definiciones de ***P*** y ***NP***, es claro que $\boldsymbol{P \subseteq NP}$, es decir, todo problema que esté en ***P*** también está en ***NP***. Sin embargo, a menos que $\boldsymbol{P = NP}$, los problemas en ***NP*** no están contenidos en ***P***.

Dentro de la clase ***NP*** hay una categoría especial, denominada ***NP-Completos***. Antes de su definición formal, se introducen conceptos importantes.

2.1.2.4.2.1 EL PROBLEMA SAT

Sean las clausulas $C_1, C_2, \dots, C_m$, tales que cada clausula consiste de operaciones de disyunción de variables de verdad $x_1, x_2, \dots, x_n$, donde $x_i = 0$ es falso y $x_i = 1$ es verdadero. Considere el problema de investigar si existen valores para las x_i de modo que $C_1 \wedge C_2 \wedge \dots \wedge C_m$ sea verdadero. A dicho problema de le denomina *Problema de Satisfacibilidad Booleana*, o simplemente SAT. Así, cada clausula debe ser verdadera para que su conjunción sea también verdadera. Dentro de cada clausula, basta con que solo una de sus componentes sea verdadera para que la clausula también sea verdadera.

- Ejemplo 15: Sea la conjunción de clausulas:

$$\mathrm{C} = (\mathrm{x}_1 \vee \mathrm{x}_2 \vee (\sim \mathrm{x}_3)) \wedge \big((\sim \mathrm{x}_1) \vee \mathrm{x}_2 \vee \mathrm{x}_4\big) \wedge (\mathrm{x}_2 \vee \mathrm{x}_3 \vee (\sim \mathrm{x}_4))$$

Donde el símbolo $\sim$ indica negación. Si se elige $x_1 = x_2 = x_3 = x_4 = 1$, entonces:

$$\mathrm{C} = (1 \vee 1 \vee (\sim 1)) \wedge \big((\sim 1) \vee 1 \vee 1\big) \wedge (1 \vee 1 \vee (\sim 1))$$

$$\mathrm{C} = (1 \vee 1 \vee 0) \wedge (0 \vee 1 \vee 1) \wedge (1 \vee 1 \vee 0)$$

Así, con el solo hecho de que existe un 1 en alguna clausula (Algún valor verdadero) ésta automáticamente es verdadera:

$$\mathrm{C} = (1) \wedge (1) \wedge (1) = 1$$

Así, se ha encontrado una asignación a las variables de verdad que hacen que la clausula general sea verdadera.

Proposición 1: $SAT \in \boldsymbol{NP}$

Prueba: Cada variable Booleana puede tomar dos posibles. Luego, si se tienen n variables, el número total de soluciones posibles es 2^n. Una MTND genera todas estas soluciones en solo n pasos (Ramificación), ya que, para cualquier conjunto de estados, genera sus ramas de manera paralela. Luego, cada origen tiene solo dos raíces, por lo que la MTND adivina el mejor camino desde cada origen hasta cada raíz. Así, el problema es resuelto en un tiempo del orden $O(n)$ por una MTND, razón por la cual SAT está en ***NP.***

2.1.2.4.2.2 EL PROBLEMA 3-SAT

Un caso especial del problema SAT es el 3-SAT, en donde cada cláusula tiene exactamente 3 literales. El Ejemplo 15 es en realidad una muestra del 3-SAT. Evidentemente, dicho problema también está contenido en ***NP***, pues se trata de un subconjunto de SAT.

2.1.2.4.2.3 REDUCIBILIDAD

Sean dos problemas de decisión π y π'. Se dice que π reduce a π' si y solo si existe un algoritmo que, en tiempo polinomial, transforma a π' en π. Esto implica que, si se tiene un algoritmo que resuelve a π, entonces el mismo algoritmo sirve para resolver a π'. Matemáticamente esto se expresa como $\pi' \preccurlyeq \pi$. Así, π es, al menos, tan difícil de resolver como π'.

2.1.2.4.3 PROBLEMAS *NP-COMPLETOS*

Sea π un problema de decisión. Se dice que π es ***NP-Completo*** si:

a) $\pi \in \boldsymbol{NP}$
b) $\forall\, \pi' \in \boldsymbol{NP}, \pi' \preccurlyeq \pi$

En el año de 1971, Stephen Cook demostró que SAT es ***NP-Completo***. De una manera empírica, la demostración puede entenderse bajo el hecho de que cualquier problema en $\boldsymbol{NP}$ puede ser transformado a un problema lógico de clausulas, que vendría a ser el SAT. Ahora se probará que 3-SAT también es ***NP-Completo***. Basta con probar que 3-SAT reduce a SAT (El cual, al ser ***NP-Completo***, reduce a todos los demás problemas ***NP***).

Proposición 2: $C_1 = (x_1 \vee \mathrm{y}) \wedge (x_1 \vee \sim \mathrm{y})$ *es verdadera si y solo si* $C_2 = x_1$ *también lo es.*
Prueba 1: (De izquierda a derecha). Supóngase que $x_1 = 0$. Por hipótesis, se necesitaría $\mathrm{y} = \sim \mathrm{y} = 1$ lo que es una contradicción. Así, si C_1 es verdadera, C_2 también lo será.
Prueba 2: (De derecha a izquierda). Trivial

Proposición 3: $C_1 = (x_1 \vee x_2 \vee \mathrm{y}) \wedge (x_1 \vee x_2 \vee \sim \mathrm{y})$ *es verdadera si y solo si* $C_2 = x_1 \vee x_2$ *también lo es.*

Prueba 1: (De izquierda a derecha). Supóngase que $x_1 = x_2 = 0$. Para verificar la hipótesis de que C_1 es verdadera, se necesita que y $=\sim$ y $= 1$, lo que claramente es una contradicción. Entonces, si C_1 es verdadera, $x_1 \vee x_2$ debe ser verdadera.
Prueba 2: (De derecha a izquierda). Para que C_2 sea verdadera (Hipótesis), por lo menos un x_i es 1, esto es, $x_1 \vee x_2$ es verdad, por lo que $x_1 \vee x_2 \vee y$ y $x_1 \vee x_2 \vee \sim$ y son clausulas verdaderas, lo que de inmediato implica que C_1 es una clausula verdadera.

Proposición 4: *Sea* $n \in N, n \geq 4$. $C_1 = (x_1 \vee x_2 \vee y_1) \wedge (\sim y_1 \vee y_2 \vee x_3) \wedge \ldots \wedge (\sim y_{n-4} \vee y_{n-3} \vee x_{n-2}) \wedge (\sim y_{n-3} \vee x_{n-1} \vee x_n)$ *es verdadera, si y solo si* $C_2 = x_1 \vee x_2 \vee \ldots \vee x_n$ *también lo es.*

Prueba 1: (De izquierda a derecha). La hipótesis es que C_1 es verdadera. En primer lugar, supóngase que ninguna x_i es 1. Entonces, para verificar la hipótesis se requiere $y_1 = y_2 = \cdots = y_{n-3} = \sim y_{n-3} = 1$, lo que conduce claramente a una contradicción ($y_{n-3} = \sim y_{n-3}$). Luego, Si C_1 es verdadero, entonces algún x_i es 1, por lo que C_2 es también verdadera.
Prueba 2: (De derecha a izquierda). Para que C_2 sea verdadera (Hipótesis), por lo menos un x_i es 1. Si $x_1 \vee x_2$ es verdadera, al hacer $y_1 = y_2 = \cdots = y_{n-3} = 0$, todas las cláusulas son verdaderas. Si $x_3 \vee x_4 \vee \ldots \vee x_{n-2}$ es verdadera, entonces al tomar $y_1 = 1, y_2 = \cdots = y_{n-3} = 0$ todas las clausulas son verdaderas. Por último, si $x_{n-1} \vee x_n$ es verdadera, entonces tomando $y_1 = y_2 = \cdots = y_{n-3} = 1$ todas las clausulas serían verdaderas. Por tanto, dada la hipótesis de que C_2 es verdadera, siempre existen valores y_k para los cuales C_1 también lo es.

Proposición 5: *El 3-SAT reduce a SAT en tiempo polinomial.*

Prueba: De las Proposiciones 1, 2 y 3, cualquier instancia del SAT puede ser transformada en una instancia del 3-SAT (o lo que es equivalente, un algoritmo que resuelva a 3-SAT también resuelve SAT). Para clausulas con un literal, basta

agregar 2 nuevos literales con sus respectivas negaciones para crear un problema 3-SAT de 4 clausulas (Proposiciones 1 y 2). Esto es claramente un orden constante ($O(c)$). Para clausulas con $n \geq 4$ literales, deben agregarse $n-3$ nuevos literales y sus negaciones

Proposición 6: $3\,SAT \in \boldsymbol{NP-Completo}$

Prueba: Como ya se mencionó anteriormente, 3-SAT está en ***NP***. Dado que SAT reduce en tiempo polinomial todos los problemas en ***NP*** (Teorema de Cook) y 3-SAT reduce en tiempo polinomial a SAT (Proposición 5), por transitividad, 3-SAT reduce a todos los problemas en ***NP*** en tiempo polinomial, por lo que 3-SAT es ***NP-Completo.***

La idea básica de probar la Proposición 6 (y por ende todas las anteriores) era crear un ambiente para demostrar más adelante que el DGAP es también ***NP-Completo.***

2.1.2.4.4 PROBLEMAS *NP-DUROS*

Sea π un problema de decisión. Se dice que π es ***NP-Duro*** si $\forall\, \pi' \in \boldsymbol{NP}, \pi' \preccurlyeq \pi$. Así, a diferencia de los problemas ***NP-Completos***, un problema en ***NP-Duro*** no necesariamente tiene porque ser ***NP***. Si los problemas ***NP-Duros*** reducen a todos los problemas en ***NP***, entonces, también reducirán a todos los problemas ***NP-Completos***, por lo que, al menos, serán igual de difíciles de resolver que estos últimos. El resumen de los diversos tipos de complejidad para un problema se muestra en el Gráfico 11.

Hasta ahora, se han estudiado solo problemas de decisión y su respectiva clasificación. Sin embargo, el estudio de las versiones de optimización es

igualmente importante, pues modifica el tipo de complejidad para muchos problemas.

- Ejemplo 16: Considere la variante decisional del TSP, descrita en el Ejemplo 14. Esta versión del problema resulto ser ***NP***. Para el problema de optimización, la MTND ya no puede adivinar si una ruta origen-raíz cumple o no una condición, y necesitaría evaluar el valor de cada posible permutación para decidir cuál es la ruta más corta, lo que imposibilita su pertenencia a ***NP.*** Sin embargo, al resolver el problema de optimización, por simple comparación se puede evaluar la condición de cualquier problema de decisión; es decir, la primera versión puede reducir a la segunda. Lo anterior implica que la versión de optimización del TSP es un problema ***NP-Duro***.

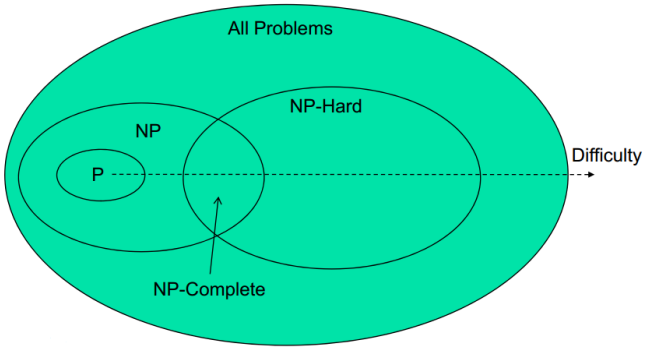

Gráfico 11: Problemas según su complejidad

2.1.3 EL PROBLEMA DE ASIGNACIÓN

El nombre "*Problema de Asignación"* fue propuesto en inicio por Votaw & Orden en el año 1952. La primera solución conocida del problema, data del año 1955, y se atribuye H.W Kuhn, en su paper seminal sobre el Método Húngaro.

El AP puede ser descrito como sigue: Se tienen n agentes y n tareas. El problema consiste en buscar una asignación tarea-agente de modo que el costo total de asignación sea mínimo. Formalmente:

$$Minimizar\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{n} X_{ij} = 1;\ \forall\, j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} X_{ij} = 1;\ \forall\, i = 1,2,\dots,n$$

$$X_{ij} = 0 \text{ ó } 1;\ \forall\, i,j$$

[5]

Donde C_{ij} es el costo de asignación de la tarea j al agente i, y X_{ij} es la variable de asignación de cada tarea a cada agente (1 si se asigna, 0 si no). La primera ecuación representa el cálculo del costo total de asignación. El segundo conjunto de ecuaciones garantiza que cada tarea sea atendida por un solo agente, mientras que el tercer conjunto de ecuaciones garantiza que cada agente atienda solo una tarea. Las variables de decisión son binarias.

Si bien el problema puede resolverse mediante un algoritmo Branch and Bound o por Planos Cortantes, H.W Kuhn ideó un algoritmo especial para resolverlo. Dicho algoritmo se denomina *Método Húngaro.* A continuación se explican sus pasos:

Sea C una matriz de $n * n$, la cual contiene los costos de que el agente i atienda la tarea j.

- ✓ Paso 1. Para cada fila i, restar el costo C_{ij} más pequeño de la fila. Sea C' la matriz resultante:

$$c'_{ij} = c_{ij} - \min\{i, c_{ij}\};\ \forall i,j \quad [6]$$

- ✓ Paso 2. Utilizar el mínimo número de líneas (horizontales o verticales) necesarias para tachar los ceros de C'. Si el número de líneas es n parar, se ha obtenido la solución óptima, de lo contrario ir al paso 3.

✓ Paso 3. Buscar el mínimo elemento no tachado en C'. Restarlo a todos los elementos no tachados y sumarlo a todos los elementos tachados por intersecciones de líneas. Luego ir al Paso 2.

- Ejemplo 17: Considere la siguiente matriz de costos asociada a un problema de asignación con 5 agentes y 5 tareas, como los muestra la Tabla 4:

10	26	13	10	22
30	19	21	11	26
11	26	11	13	23
26	23	29	16	11
26	19	24	24	21

Tabla 4: Costos del Problema de Asignación del Ejemplo 17

En primer lugar, a cada elemento se le resta el mínimo de su fila (10,11,11,11 y 19, respectivamente):

0	16	3	0	12
19	8	10	0	15
0	15	0	2	12
15	12	18	5	0
7	0	5	5	2

Fácilmente puede verse que el número mínimo de líneas para cubrir todos los ceros resultantes es de 5:

0	16	3	0	12
19	8	10	0	15
0	15	0	2	12
15	12	18	5	0
7	0	5	5	2

Como el número de líneas equivale a n (5), se ha encontrado la solución óptima. De la última matriz se tiene que $X_{11} = X_{24} = X_{33} = X_{45} = X_{52} = 1$, con $Z = 10 + 11 + 11 + 11 + 19 = 62$.

2.1.4 ALGORITMOS GENÉTICOS

Dado que los problemas ***NP***, ***NP-Completos*** y ***NP-Duros*** son difíciles de resolver de manera óptima (no se conocen algoritmos polinomiales para ello), frecuentemente se recurre a métodos de aproximación para tratarlos; es decir, se emplean procedimientos que brinden una respuesta satisfactoria en un tiempo razonable. Entre dichos procedimientos pueden citarse las heurísticas y las metaheurísticas. Las primeras corresponden a algoritmos que dejan de lado la noción de optimalidad, e intentan encontrar buenas soluciones en tiempos de cómputo razonable, basándose en las características propias del problema. Una heurística se diferencia de cualquier otro tipo de algoritmo en que, no tienen una garantía plena de su respuesta, ya sea en solución, en tiempo o ambas.

Por su parte, las metaheurísticas son procedimientos algorítmicos genéricos y abstractos cuya parametrización depende del problema que se vaya a resolver y de los requerimientos del usuario, permitiendo aplicar la misma estructura general a diversos problemas. Entre las metaheurísticas más populares en el campo de la optimización combinatoria, se pueden citar Recocido Simulado, Colonias de Hormigas, Búsqueda Tabú y Algoritmos Genéticos. Éstos últimos son de especial interés en el desarrollo de este proyecto, ya que, han probado ser bastante buenos (Tanto en calidad como en eficiencia) para resolver muchos problemas complejos.

Los Algoritmos Genéticos son algoritmos estocásticos y metaheurísticos, inspirados en el proceso evolutivo de los seres vivos en la naturaleza. Es decir, estos simulan los cambios genéticos que experimentan los seres vivos durante el proceso de adaptabilidad a lo largo de diversas generaciones de su especie. Fueron considerados por primera vez por John Holland en el año de 1970.

En la naturaleza, según la teoría de selección natural, solo los individuos mejor adaptados al medio sobreviven. Estos se superponen a los individuos más débiles (hablando de una especie en particular) teniendo mayor probabilidad de aparearse y reproducirse. Los nuevos individuos en la población, tendrán información genética de los padres (Leyes de Mendel), por lo que heredarán sus habilidades instintivas para adaptarse y sobrevivir a su entorno. Si estos nuevos individuos afrontan entornos distintos, se adaptan a los mismos modificando la información genética transmitida a su descendencia. Este proceso de adaptación y reproducción continúa a lo largo de la existencia de la especie. Pero, la pregunta que aquí resulta es: ¿Cómo ocurre esto en los seres vivos a nivel celular?

Internamente, todos los seres vivos reaccionan de manera dinámica a los cambios que se dan en el entorno; por ejemplo, a lo largo de la historia, los osos polares desarrollaron una piel gruesa y un alto contenido de grasa corporal, como respuesta a las bajas temperaturas a las que estaban sometidos. Las características como color del pelaje, estatura promedio, etc., se denominan *fenotipo;* sin embargo, de manera interna, el fenotipo es codificado en un lenguaje especial, denominado *genotipo*, que puede decirse es el código bajo el cual se almacenan los rasgos de las especies a nivel celular. Dicho de otro modo, el fenotipo es la expresión del genotipo. La información en cuestión se almacena en los *genes,* los cuales son cadenas cortas de *ADN*. Los genes a su vez forman lo que se denomina *cromosomas*.

Durante la formación de una nueva vida, ocurre un cruzamiento de cromosomas (provenientes del padre y de la madre), con lo cual se comparte información genética. El nuevo individuo, toma características genéticas de ambos padres (aunque algunas no se manifiesten en el fenotipo).

La evolución ha probado que, a lo largo de las generaciones, las características de las especies han ido mejorando debido al cruzamiento de la información genética de sus predecesores. En general, el proceso de transformación genética se da mediante tres mecanismos, los cuales son, en su orden, selección, cruzamiento y mutación. El primero, radica en un principio muy importante en la teoría de

selección natural: "Sólo los individuos mas fuertes sobreviven", por lo que solo los más "aptos" podrán reproducirse con mayor probabilidad y así poder transmitir su información genética a la siguiente generación. El segundo, corresponde al proceso en el que los padres comparten información genética, la cual es transmitida a su descendiente. Por último, la mutación corresponde a un cambio en la información genética que resulta del cruzamiento, dando lugar a la aparición de una o varias características nuevas en la descendencia.

Los algoritmos genéticos corresponden a una imitación de los procesos evolutivos en la naturaleza, que se dan a través de la selección, cruzamiento y mutación genética, aplicados a problemas de optimización de la vida real. Aunque John Holland fue el primero en introducir el concepto de algoritmo genético aplicado a la optimización, hubo precedentes en lo que respecta a la computación evolutiva. Científicos de los años 60, tales como A.J Owens y M.J Walsh introdujeron el concepto de programación evolutiva. En sus estudios, solo utilizaban el mecanismo de mutación para producir nuevas generaciones en la población. Estas y otras ideas comenzaron a dar vida a lo que hoy se conoce como inteligencia artificial, que consiste en programas de cómputo que retienen los comportamientos de un sistema real. A mediados de los 70 David Goldberg, estudiante de Holland, fue el primero en aplicar a un problema industrial un algoritmo genético.

Un problema de optimización puede verse desde el punto de vista genético como sigue: Los individuos (soluciones al problema) están representados no por su fenotipo, sino por su genotipo mediante cromosomas (vector de variables), donde cada cromosoma está constituido por cierta cantidad de genes (variables). Cada individuo tiene cierto grado de *aptitud,* la cual determina el grado de adaptación al entorno (valor de la función objetivo evaluada en la solución). Por selección natural, los individuos mejor adaptados al medio tienen mayor probabilidad de reproducirse (soluciones con mejores valores objetivo). Después de la reproducción (cruzamiento de las soluciones seleccionadas) se genera una nueva población, la cual puede experimentar mutaciones en su información genética (variación en la solución). El

proceso se repite hasta que se alcance cierto número de generaciones o cuando la aptitud de los individuos es lo suficientemente aceptable. Debe tenerse en cuenta que la aptitud dependerá del tipo de optimización; si el objetivo es maximizar una función, entonces la aptitud corresponde al valor de la dicha función, mientras que si lo que se persigue es la minimización, entonces se usa el recíproco de la función.
El pseudocódigo del algoritmo genético se ilustra en el Gráfico 12. Los algoritmos genéticos se diferencian de las herramientas de optimización convencionales en varios puntos:

- No necesitan información previa de las características del problema que están resolviendo.
- Utilizan varias soluciones del problema en una misma iteración.
- Usan reglas probabilísticas en su proceso.
- No suelen trabajar con variables en base 10. En lugar de ello, trabaja con variables codificadas que permiten emular de manera más acertada los procesos genéticos.

Gráfico 12: Pseudocódigo del algoritmo genético

A continuación se mencionan los elementos más importantes a la hora de construir un algoritmo genético.

2.1.4.1 SOLUCIONES INICIALES

Una de las peculiaridades de los algoritmos genéticos es que trabajan con varias soluciones a la vez, las cuales constituyen la población inicial. En general, el número de soluciones iniciales (tamaño de la población) es un parámetro que el diseñador y analista del algoritmo debe escoger a su conveniencia.

La forma de obtener la población inicial puede darse de varios modos; se pueden generar de manera aleatoria las soluciones iniciales, o bien, utilizar alguna heurística inicial. Esto último puede impactar de manera significativa la calidad del algoritmo, ya que si se trabaja con soluciones aleatorias que estén muy lejos del óptimo, la tasa de convergencia puede llegar a ser lenta.

2.1.4.2 CODIFICACIÓN

Antes de establecer la estructura de los mecanismos evolutivos, debe seleccionarse de manera correcta de codificar las variables. Noor (2007) estableció diversos tipos de codificación dependiendo del tipo de cruzamiento utilizado. En general, existen cuatro tipos de codificación: Binaria, por permutación, por valor real y por árbol.

2.1.4.2.1 CODIFICACIÓN BINARIA

Es la codificación más empleada en la mayoría de aplicaciones de algoritmos genéticos. Para el caso en que se trabaje en base dos, hacer la codificación es bastante simple (Se asumen números enteros). En primer lugar, se establecerán un conjunto de reglas necesarias para codificar cualquier valor en base 10.

Proposición 7: *Sea* $S_n = 1 + 2 + 2^2 + \cdots + 2^{n-1}$. *Entonces* $S_n = 2^n - 1$

Prueba: Se tiene que:

$$S_n = S_{n-1} + 2^{n-1}$$

$$2^{n-1} = S_{n-1} + 1$$

Restando la segunda ecuación de la primera:

$$S_n - 2^{n-1} = 2^{n-1} - 1 \rightarrow S_n = 2 * 2^{n-1} - 1$$

$$\rightarrow S_n = 2^n - 1$$

Proposición 8: *Sea* x *un número en base 10. El número de bits necesarios para escribir* x *en base 2 es* $n \geq \left\lceil \frac{Ln(x+1)}{Ln(2)} \right\rceil$

.

Prueba: En base 2, x puede ser escrito como $y_1 y_2 \ldots y_n$, donde $y_i = 0$ ó $1 \; \forall \; i$. Al pasar el número a base 10, se tiene:

$$x = \sum_{i=0}^{n-1} y_i (2)^i \leq \sum_{i=0}^{n-1} (2)^i = 2^n - 1 \; (Proposición\; 7)$$

Luego:

$$2^n - 1 \geq x \rightarrow 2^n \geq x + 1 \rightarrow nLn(2) \geq Ln(x+1)$$

$$\rightarrow n \geq \frac{Ln(x+1)}{Ln(2)}$$

Como el entero más pequeño que es mayor o igual a $\frac{Ln(x+1)}{Ln(2)}$ es $\left\lceil \frac{Ln(x+1)}{Ln(2)} \right\rceil$ entonces:

$$\rightarrow n \geq \left\lceil \frac{Ln(x+1)}{Ln(2)} \right\rceil.$$

Gracias a lo anterior, se pueden establecer los pasos para la codificación:

$Paso\ 1: Sea\ x\ el\ número\ a\ codificar. El\ número\ de\ bits\ necesarios\ para\ su\ representación\ en\ binario\ viene\ dado\ por\ n = \left\lceil \frac{Ln(x+1)}{Ln(2)} \right\rceil. hacer\ i = 0.$

$Paso\ 2: Si\ i = n\ terminar, sino\ hacer\ bit(i) = x\ mod\ 2, x = \left\lfloor \frac{x}{2} \right\rfloor, i = i - 1\ y\ repetir\ paso\ 2.$

- Ejemplo 18: Se desea resolver el siguiente problema de optimización:

$$max\ f(x_1, x_2, x_3) = {x_1}^2 + 2\,x_1x_2 - 2x_2x_3 + {x_3}^2$$
$$x_i \in Z, 0 \leq x_i \leq 19;\ \forall\ i = 1,2,3.$$

Si se quisiera utilizar una codificación binaria para resolver el problema mediante algoritmos genéticos, debe tenerse en cuenta en primer lugar el valor máximo que puede tomar cada variable. El número mínimo de bits será de $n = \left\lfloor \frac{Ln(19+1)}{Ln\,(2)} \right\rfloor = 5$. La Tabla 5 resume los resultados de aplicar los pasos de codificación al número 19:

I	X	$x\ mod\ 2$	$\left\lfloor \frac{x}{2} \right\rfloor$
4	19	1	9
3	9	1	4
2	4	0	2
1	2	0	1
0	1	1	0

Tabla 5: Pasos para codificar el número 19 a base 2

Luego, la representación del 19 en base 2 es 11001. La decodificación es bastante simple:

$$\sum_{i=0}^{n-1} y_i(2)^i = 1(2)^0 + 1(2)^1 + 0(2)^2 + 0(2)^3 + 1(2)^4 = 1 + 2 + 16 = 19$$

Supóngase que se tiene una solución al problema, donde $x_1 = 2,\ x_2 = 10, x_3 = 13$. Puede verificarse que en código binario $x_1 = 01000, x_2 = 01001, x_3 = 10110$.
Un cromosoma está conformado por las variables del problema. La representación de la solución mencionada se ilustra en el Gráfico 13.

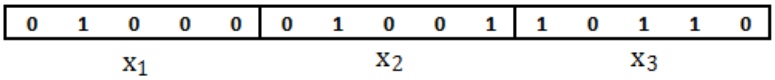

Gráfico 13: Codificación Ejemplo 18.

2.1.4.2.2 CODIFICACIÓN POR PERMUTACIÓN

Una permutación es un arreglo de elementos de una lista inicial, donde cada elemento aparece una y solo una vez. El cromosoma tendrá entonces tantos bits como elementos tenga la lista, y en cada bit (gen) irá el valor de lista correspondiente a cada posición.

- Ejemplo 19**:** Considere la lista $\boldsymbol{L} = [\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}]$. Una permutación de dicha lista puede ser $\boldsymbol{\pi} = [\mathbf{3}, \mathbf{4}, \mathbf{2}, \mathbf{8}, \mathbf{1}, \mathbf{5}, \mathbf{7}, \mathbf{6}]$. En código por permutación, el cromosoma vendría dado por **3 4 2 8 1 5 7 6** , donde cada posición viene siendo un gen.

2.1.4.2.3 CODIFICACIÓN POR VALOR

Utiliza el valor real de las variables en el cromosoma. Del Ejemplo 2.18, el cromosoma para $x_1 = 2$, $x_2 = 10, x_3 = 13$ es precisamente $2\ 10\ 13$.

2.1.4.2.4 CODIFICACIÓN POR ÁRBOL

Esta codificación resulta útil para problemas donde las soluciones pueden representarse mediante árboles. No suele aplicarse con frecuencia a problemas de optimización.

2.1.4.3 SELECCIÓN

En cada generación del algoritmo, se deben seleccionar los individuos de la población que se van a cruzar. Como ya se mencionó anteriormente, la elección de un individuo (solución) dependerá de su aptitud (valor objetivo); sin embargo, existen muchas formas de llevar a cabo este mecanismo durante la ejecución del algoritmo.

La probabilidad de que un individuo sea seleccionado para cruzarse, es directamente proporcional a su aptitud dentro de la población. Sea P el número de individuos que componen la población, y sea a_i la aptitud del individuo i. Si se considera $A_P = \sum a_i$ como la aptitud conjunta de la población, la aptitud relativa del individuo i, denominada $\mathrm{r_i}$, viene dada por:

$$r_i = \frac{a_1}{A_P} = \frac{a_i}{\sum a_i} \quad [7]$$

La aptitud relativa de un individuo también puede ser visto como la probabilidad que tiene de ser seleccionado. Así, los individuos se seleccionan según su aptitud relativa.

En términos computacionales, debe diseñarse un experimento que permita simular el proceso de selección mediante números pseudoaleatorios. Para ello, se toma un orden cualquiera de los individuos, y a cada uno se le calcula la probabilidad acumulada. Se genera un valor pseudoaleatorio y el primer individuo cuya probabilidad acumulada sea mayor a igual al valor generado es escogido para el proceso de cruzamiento. A éste procedimiento se le denomina *ruleta*. Un ejemplo se muestra en el Gráfico 14, donde se tienen 4 individuos en la población. Cada porción en la ruleta corresponde a la probabilidad de que el individuo sea seleccionado.

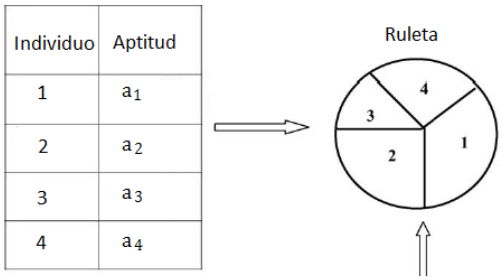

Gráfico 14: Ejemplo de selección por ruleta

Hay diversas variantes para la selección por ruleta, como se explicará a continuación:

2.1.4.3.1 SELECCIÓN POR RULETA CON REEMPLAZO

Mediante esta variante, un individuo puede ser elegido un máximo de P veces, debido a que las probabilidades en la ruleta no cambian durante la selección.

- Ejemplo 20: Considere el problema expuesto en el Ejemplo 18, donde se da la solución inicial $x_1 = 2,\ x_2 = 10, x_3 = 13$. Suponga además otras tres soluciones: $x_1 = 3,\ x_2 = 1, x_3 = 0;\ x_1 = 10,\ x_2 = 5, x_3 = 1$ y $x_1 = 3,\ x_2 = 3, x_3 = 3$. Se tendría entonces una población con 4 individuos, cada uno con aptitud 83, 15, 191 y 18, respectivamente. Para cada individuo, la probabilidad de elección es:

$$r_1 = \frac{83}{83 + 15 + 191 + 18} = \frac{83}{307} = 0{,}27$$

$$r_2 = \frac{15}{307} = 0{,}049$$

$$r_3 = \frac{191}{307} = 0{,}622$$

$$r_4 = \frac{18}{307} = 0{,}059$$

La Tabla 6 resume los anteriores resultados, así como el cálculo de la probabilidad acumulada para cada individuo. Si para la ruleta se genera un número pseudoaleatorio igual a 0,45, entonces el individuo seleccionado para el cruzamiento es el 3; si el número generado es 0,18, el individuo seleccionado es el 1. Según el método bajo estudio, un mismo individuo puede ser seleccionado hasta cuatro veces (tamaño de la población). La Tabla 7 muestra la generación de los cuatro números en cuestión mediante calculadora, así como el individuo seleccionado en cada caso. Puede verse que, tanto el individuo 1 como el individuo 3 fueron seleccionados tres veces; esto no es coincidencia, pues estos son precisamente los individuos con mayor probabilidad de elección.

Proposición 9: *El número de pasos de la selección por ruleta con reemplazo es de orden* $O(P^2)$.

Demostración: La simulación de la ruleta consiste en generar un número pseudoaleatorio y buscar el individuo que tenga la primera probabilidad acumulada mayor o igual a dicho número. En peor de los casos, el valor generado corresponderá a una asignación del último individuo, con lo que se harían P

comparaciones. Como esto se repetiría P veces, el total de comparaciones sería de P^2, es decir, el número de pasos de esta selección es de orden máximo $O(P^2)$.

$Individuo$	$Probabilidad\ de\ Elección$	$Probabilidad\ Acumulada$
1	0,27	0,27
2	0,049	0,319
3	0,622	0,941
4	0,059	1

Tabla 6: Cálculo de probabilidades para los individuos del Ejemplo 20

pseudoaleatorio	Individuo
0,221	**1**
0,652	**3**
0,912	**3**
0,196	**1**

Tabla 7: Selección de individuos en el Ejemplo 20

2.1.4.3.2 SELECCIÓN POR RULETA CON REEMPLAZO PARCIAL

La selección por ruleta con reemplazo puede presentar un problema serio. Al no haber límite sobre la cantidad de veces que un individuo puede ser seleccionado para el cruzamiento (los de mayor probabilidad), solo se cruzaran los individuos mas dominantes, incrementando la probabilidad de que la soluciones converjan de manera prematura a óptimos locales. Una manera de contrarrestar este inconveniente es mediante el uso de la selección por ruleta con reemplazo parcial.
Éste método establece una cota superior sobre el número máximo de veces que un individuo puede ser seleccionado; dicha cota viene dada por $\lceil nr_i \rceil$. Si durante la

selección se escoge a un individuo que llegó a su límite, la ruleta debe volverse a girar hasta seleccionar un individuo factible. Resulta preferible sacar un individuo que llegó a su máximo de elecciones de la lista de candidatos, recalcular todas las probabilidades y volver a girar la ruleta, teniendo en cuenta el número de asignaciones anteriores.

- Ejemplo 21: Considere la situación expuesta en el Ejemplo 20. El número límite de elecciones para cada individuo se muestra en la Tabla 8. Si el individuo 3 ha sido seleccionado 3 veces, y el próximo número pseudoaleatorio resulta ser 0,78 (ver Tabla 6), debe generarse un nuevo número, ya que, no se puede seleccionar más de tres veces al individuo en cuestión.

$Individuo$	$Máximo\ de\ elecciones$
1	$\lceil 0{,}27(4) \rceil = 2$
2	$\lceil 0{,}049(4) \rceil = 1$
3	$\lceil 0{,}622(4) \rceil = 3$
4	$\lceil 0{,}059(4) \rceil = 1$

Tabla 8: Número máximo de elecciones para los individuos del Ejemplo 20

Proposición 10: *El número de pasos de la selección por ruleta con reemplazo parcial es de orden* $O(P^2)$*.*

Prueba: En el peor caso, un individuo podrá seleccionarse máximo una vez, lo que lleva al recalculo de todas las probabilidades en $2(P-i)$ pasos, siendo i el individuo en cuestión. Para cada elección, se hacen máximo $P-i+1$ pasos hasta encontrar la probabilidad acumulada que cumpla la condición para el valor pseudoaleatorio. El número de pasos para reajustes vendría a ser:

$$Reajustes = 2(P-1) + 2(P-2) + \cdots + 2 + 1 = 2\left(P^2 - \frac{P(P+1)}{2}\right) = P(P-1)$$

El número de búsquedas efectuadas se calcula como sigue:

$$Búsquedas = P + (P-1) + (P-2) + \cdots + 1 = \frac{P(P+1)}{2}$$

Así, el total de pasos empleados por éste método de selección viene dado por:

$$Total\ Pasos = P(P-1) + \frac{P(P+1)}{2} \leq P^2 + \frac{P(2P)}{2} = 2P^2 = O(P^2)$$

2.1.4.3.3 SELECCIÓN POR RULETA DEL RESTO CON REEMPLAZO

A diferencia del método anteriormente descrito, en éste se establece una cota inferior para el número de veces que es seleccionado un individuo. La cota inferior viene dada por $\lfloor \mathrm{nr_i} \rfloor$. Una vez asignado el número de veces mínimo que cada individuo debe aparecer, se recalcula la probabilidad de elección como $r'_i = (nr_i - \lfloor nr_i \rfloor)/(n - \sum \lfloor nr_i \rfloor)$. Después, se usa la selección por ruleta con reemplazo con las nuevas probabilidades.

- Ejemplo 22: Se retoma nuevamente el Ejemplo 20. La Tabla 9 ilustra el cálculo del número mínimo de veces que cada individuo debe ser seleccionado. De la tabla, se tiene que solo debe girarse la nueva ruleta una vez, pues automáticamente se han seleccionado tres individuos.

Individuo	*Mínimo de elecciones*
1	$\lfloor 0{,}27(4) \rfloor = 1$
2	$\lfloor 0{,}049(4) \rfloor = 0$
3	$\lfloor 0{,}622(4) \rfloor = 2$
4	$\lfloor 0{,}059(4) \rfloor = 0$

Tabla 9: Número mínimo de elecciones para los individuos del Ejemplo 20

Para recalcular las probabilidades de elección, el denominador necesario seria $4-(1+0+2+0)=1$. El cálculo de las nuevas probabilidades individuales y acumuladas se representa en la Tabla 10.

$Individuo$	$Nueva\ Probabilidad\ de\ Elección$	$Probabilidad\ Acumulada$
1	$\frac{0{,}27(4)-\lfloor 0{,}27(4)\rfloor}{1}=0{,}08$	$0{,}08$
2	$\frac{0{,}049(4)-\lfloor 0{,}049(4)\rfloor}{1}=0{,}196$	$0{,}276$
3	$\frac{0{,}622(4)-\lfloor 0{,}622(4)\rfloor}{1}=0{,}488$	$0{,}764$
4	$\frac{0{,}059(4)-\lfloor 0{,}059(4)\rfloor}{1}=0{,}236$	1

Tabla 10: Cálculo de nuevas probabilidades según selección por ruleta del resto con reemplazo

Proposición 11: *El número de pasos de la selección por ruleta del resto con reemplazo es de orden* $O(P^2)$*.*

Prueba: Este método encuentra en número mínimo de selecciones para todos los individuos en P pasos. Sea k el número de selecciones efectuadas. El resto de las $P-k$ selecciones se hacer por ruleta con reemplazo. Como el número de pasos sería proporcional a $(P-k)^2$, en el peor caso, k sería cero, por lo que el método tiene un orden máximo $O(P^2)$.

2.1.4.3.4 SELECCIÓN POR RULETA DEL RESTO SIN REEMPLAZO

En este método resulta ser una combinación de la selección por ruleta con reemplazo parcial y la selección por ruleta del resto con reemplazo. Así, una vez calculado el número mínimo de selecciones por individuo, se recalculan las

probabilidades de selección, usando en el nuevo conjunto selección por ruleta con reemplazo parcial.

Todos los métodos anteriores pueden presentar un inconveniente. Si la mayoría de los individuos tienen aptitudes muy cercanas, sus porciones en la ruleta serán muy similares, lo que tendrían probabilidades muy parecidas de ser seleccionados. Lo anterior no es acorde al principio de selección natural.
Existe un método denominado *Escalado,* que permite hacer una transformación en las aptitudes iniciales, de modo que la varianza se incremente. Otros métodos son utilizados para contrarrestar el problema de poca variabilidad, pero no son de interés para la ejecución de este proyecto.

2.1.4.4 CRUZAMIENTO

El cruzamiento es considerado el operador genético más importante. Mediante este, los individuos intercambian bloques de alelos, intercambiando al mismo tiempo información genética. Existen diversos tipos de operadores de cruzamiento, entre los cuales pueden mencionarse el cruzamiento en un solo punto, el cruzamiento multipunto, cruzamiento uniforme y cruzamiento aritmético. A continuación se detalla cada uno de ellos.

2.1.4.4.1 CRUZAMIENTO EN UN PUNTO

Considérense dos individuos (cromosomas) cada uno con L genes; el número de puntos posibles para el cruzamiento será $L-1$. Entonces, se selecciona aleatoriamente uno de los $L-1$ puntos, a partir del cual se efectuará el cruce.

- Ejemplo 23: Considere los cromosomas 0 1 0 1 1 0 y 1 1 0 0 0 1. Supóngase que el cruzamiento se va a hacer en el tercer punto. El resultado del cruzamiento estaría dado por los descendientes 0 1 0 0 0 1 y 1 1 0 1 1 0.

2.1.4.4.2 CRUZAMIENTO MULTIPUNTO

Para este caso, se seleccionan de manera aleatoria L' puntos de cruce, con $L' \leq L - 1$, intercambiando solo los semi bloques que queden antes de una posición de cruzamiento impar. Note que el cruce en un punto es un caso particular del cruce en un punto, con $L' = 1$.

- Ejemplo 24: Considere los cromosomas 1 0 1 0 1 0 1 0 1 0 y 0 0 0 0 0 1 1 1 1 1. Si se escogen aleatoriamente los puntos 3, 6 y 9 para el cruzamiento, los individuos resultantes serán 1 0 1 0 0 1 1 0 1 1 y 0 0 0 0 1 0 1 1 1 0 . Debe notarse que el punto tres es la primera posición de cruce (uno es impar), 6 la segunda posición (dos es par) y 9 la tercera posición (impar).

2.1.4.4.3 CRUZAMIENTO UNIFORME

Se genera aleatoriamente una cadena de ceros y unos, de longitud L; dicha cadena se denomina máscara. Una vez generada la cadena, el procedimiento es el siguiente: Se recorren todas las posiciones de la máscara. Si el elemento i de la máscara es un uno, entonces el elemento i del padre 1 pasa a ser el elemento i del hijo 1, y el elemento i del padre 2 pasa a ser el elemento i del hijo 2. Si el elemento de la máscara resulta ser un cero, entonces el elemento i del padre 1 pasa a ser el elemento i del hijo 2, y el elemento i del padre 2 pasa a ser el elemento i del hijo 1. Así, para cada descendiente este proceso se ejecuta con una complejidad máxima

$O(L)$, por lo que para toda la población P esto se realiza con complejidad máxima $O(LP)$.

- Ejemplo 25: Considere los cromosomas 1 0 1 0 1 0 1 0 1 0 y 0 0 0 0 0 1 1 1 1 1 definidos en el Ejemplo 24. Si se escogen aleatoriamente los puntos 3, 6 y 9 para el cruzamiento. Suponga que se generó la máscara 0 0 0 1 0 1 0 0 1 1. Luego, los dos descendientes vienen a ser:
  - Hijo 1: 0 0 0 0 0 0 1 1 1 0.
  - Hijo 2: 1 0 1 0 1 1 1 0 1 1 .

2.1.4.4.4 CRUZAMIENTO ARITMÉTICO

Este es un cruzamiento especial para los casos de codificación por valor. El valor de cada gen en un hijo se obtiene como una combinación lineal de los genes del padre. Sea w un número aleatorio entre 0 y 1. El cruzamiento aritmético se define matemáticamente como sigue:

$$Hijo\ 1 = w[Cromosoma\ \ Padre\ 1] + (1 - w)[Cromosoma\ \ Padre\ 2] \quad [8]$$

$$Hijo\ 2 = w[Cromosoma\ \ Padre\ 2] + (1 - w)[Cromosoma\ \ Padre\ 1] \quad [9]$$

- Ejemplo 26: Sean los cromosomas $3\ 4\ 6\ 2\ 0$ y $2\ 3\ 5\ 7\ 2$. Si $w = 0.4$ entonces los descendientes al aplicar cruzamiento aritmético vendrían dados por:

$$Hijo\ 1 = 0.4[3\ 4\ 6\ 2\ 0\ ] + 0.6[2\ 3\ 5\ 7\ 2\ ] = 2.4\ \ 3.4\ \ 5.4\ \ 5\ \ 1.2$$

$$Hijo\ 1 = 0.4[2\ 3\ 5\ 7\ 2\ ] + 0.6[3\ 4\ 6\ 2\ 0\ ] = 2.6\ \ 3.6\ \ 5.6\ \ 4\ \ 0.8$$

2.1.4.5 MUTACIÓN

La mutación es el último de los operadores genéticos. Consiste en la alteración aleatoria de uno o varios genes en la descendencia después del cruzamiento, de manera que se tenga posibilidad de que aparezcan nuevas características que

posiblemente se encuentren ausente en los padres. En términos del problema, esto implicaría moverse a una nueva región del espacio de soluciones, lo que ayudaría a evadir posibles óptimos locales.

Al igual que ocurre en la naturaleza, la probabilidad de que un gen mute es muy pequeña, ya que es poco probable que un individuo tenga una característica que no se encuentre presente en el genotipo de sus padres.

Existen diversas técnicas para llevar a cabo el operador mutación. Michalewicz, (1994) , Michalewicz (1996) y Choy & Sanctuary (1998) propusieron diversos métodos que se ajustan a distintos tipos de problemas. En este proyecto no se detallan dichas metodologías pues resultan ser de poco interés.

2.1.4.6 EL TEOREMA DE LOS ESQUEMAS

En muchas ramas de la ciencia se usan los algoritmos genéticos para resolver diversos tipos de problemas. Sin embargo, pocos indagan sobre el por qué estos algoritmos son eficientes. La argumentación matemática que permite verificar por qué funcionan los algoritmos genéticos se denomina Teorema de los Esquemas.

Un esquema puede verse como un conjunto de genes presente en diversos individuos; por ejemplo, el conjunto $0 * * 1 *$ es un esquema para un cromosoma de cinco genes, en el que la primera posición siempre es 0 y la cuarta 1, es decir, es un esquema de longitud 4 (la longitud se calcula teniendo en cuenta el primer y último valor numérico del esquema). Así, en los cromosomas $0\,0\,1\,1\,1$ y $0\,1\,0\,1\,0$ se encuentra presente dicho esquema. La cantidad de números fijos en un esquema determina su *orden*. Para el ejemplo anterior, el orden del esquema es 2.

Sea S un esquema de interés, con longitud $\delta(S)$ y orden $o(S)$. Del operador de selección, se sabe que la probabilidad de que un individuo sea seleccionado es directamente proporcional a su aptitud, dada ésta por la ecuación $r_i = \frac{a_i}{\sum a_i}$. Sea además η_t el número de individuos con el esquema S presentes en la generación t, siendo $\eta_t \leq P$. Entonces, el número esperado se individuos con el esquema S seleccionados para cruzamiento es:

$$\eta_t(Cruzamiento) = P \sum_{i \, \epsilon \, S} \frac{a_i}{A_P} \quad [10]$$

Definiendo la aptitud promedio del esquema S como $\overline{A_S} = \frac{\sum_{i \, \epsilon \, S} a_i}{\eta_t}$ y la aptitud promedio de la población como $\overline{A_P} = \frac{A_P}{P}$:

$$\eta_t(Cruzamiento) = P \sum_{i \, \epsilon \, S} \frac{a_i}{A_P} = \frac{P}{A_P} \sum_{i \, \epsilon \, S} a_i = \frac{\eta_t \overline{A_S}}{\overline{A_P}} \quad [11]$$

Como puede verse [11], si un esquema tiene una aptitud promedio por encima de la aptitud promedio de la población, la cantidad esperada de sus individuos seleccionado para cruzamiento crece con respecto a la cantidad inicial presente en la población.

Resulta de interés ver la cantidad de individuos de S que pasa a la generación $t + 1$. Para que esto se dé, los individuos pertenecientes al esquema deben sobrevivir tanto al cruzamiento como a la mutación (se asume cruzamiento en un punto). La probabilidad de que cualquier punto del esquema sea seleccionado para el cruzamiento es $\frac{\delta(S)}{L-1}$. Luego de ser seleccionado, la probabilidad de que el cruce se haga efectivo viene dada por $p_c \frac{\delta(S)}{L-1}$, siendo p_c la probabilidad condicionada de cruce. Luego, la probabilidad de que no haya cruzamiento en ninguno de los puntos del esquema es $1 - p_c \frac{\delta(S)}{L-1}$. La mutación puede ocurrir en cualquier cantidad de puntos del esquema. Así, la única forma en que se sobreviva a la mutación es que ninguno de los genes del esquema mute, lo que equivale a $(1 - p_m)^{o(S)}$, siendo p_m la probabilidad de que un gen cualquiera mute. Por tanto, la cantidad esperada de individuos con el esquema S en la generación $t + 1$ es:

$$E(\eta_{t+1}) = \frac{\eta_t \overline{A_S}}{\overline{A_P}} \left(1 - p_c \frac{\delta(S)}{L-1}\right) (1 - p_m)^{o(S)} \quad [12]$$

Como se aprecia en la última ecuación entre menor sean la longitud y el orden del esquema, mayor será su cantidad de individuos en las siguientes generaciones. Formalmente, el teorema de los esquemas dice: *Esquemas de bajo orden, de corta longitud y con aptitud por encima de la media tienen un número exponencialmente creciente de individuos en generaciones siguientes.*

Para hacer la prueba formal del teorema, se utiliza el resultado expuesto en [12] y la siguiente ecuación de recurrencia:

$$E(\eta_{t+1}) = E(\eta_t)\frac{\overline{A_S}}{\overline{A_P}}\left(1 - p_c\frac{\delta(S)}{L-1}\right)(1-p_m)^{o(S)} \quad [13]$$

Siendo $E(\eta_0) = \eta_0$ un valor constante y conocido (resultante de la población inicial). Haciendo $\frac{\overline{A_S}}{\overline{A_P}}\left(1 - p_c\frac{\delta(S)}{L-1}\right)(1-p_m)^{o(S)} = k$:

$E(\eta_{t+1}) = E(\eta_t)k = (E(\eta_{t-1})k)k = E(\eta_{t-1})k^2 = E(\eta_{t-2})k^3 = \cdots = E(\eta_0)k^{t+1}$
$= \eta_0 k^{t+1}$

$$E(\eta_{t+1}) = \eta_0\left(\frac{\overline{A_S}}{\overline{A_P}}\left(1 - p_c\frac{\delta(S)}{L-1}\right)(1-p_m)^{o(S)}\right)^{t+1} \quad [14]$$

Por hipótesis y $\frac{\overline{A_S}}{\overline{A_P}}\left(1 - p_c\frac{\delta(S)}{L-1}\right)(1-p_m)^{o(S)} > 1$ (las probabilidades de no cruzamiento y no mutación son lo suficientemente cercanas a uno, y la aptitud del esquema es lo suficientemente superior a uno), razón por la que η_t crece exponencialmente.

2.2 ESTADO DEL ARTE

No hay investigaciones que hablen explícitamente del problema aquí tratado. Sin embargo, se señalarán las variantes del Problema de Asignación que se encuentran disponibles en la literatura, así como los métodos usados para su resolución (especialmente para el GAP).

2.2.1 PROBLEMA DE ASIGNACIÓN Y SUS VARIANTES

Los problemas de asignación son un caso especial de la programación matemática, con un alto nivel de aplicabilidad a situaciones de la vida real. El problema consiste en asignar un conjunto de tareas (Trabajos, clientes, piezas), a un conjunto de agentes (Celdas de manufactura, centros de distribución, máquinas), de modo que se optimice una función o un conjunto de funciones.

Muchas son las variantes y soluciones que se han propuesto al Problema de Asignación. Entre los algoritmos más usados se tienen desde la Programación Meta aplicada al Problema de Asignación Clásico (Jahanshahloo & Afzalinejad, 2008) hasta algoritmos genéticos aplicados a la variante de Asignación de Proyectos (Harper, de Senna, Vieira, & Shahani, 2005).

Una revisión de la literatura completa sobre las distintas variantes del Problema de Asignación es presentada por Pentico (2007) en donde pueden encontrarse variantes tales como el Problema de Asignación con Calificación de Agentes, Problema de Asignación con Cardinalidad, Problema de Asignación Cuello de Botella (BAP, por sus siglas en inglés), Problema de Asignación Balanceado, Problema de Asignación con Mínima Desviación, El Problema Lexicográfico de Cuello de Botella, Problema de Asignación k Sigma, Problema de Semi Asignación, Problema de Asignación Categorizado, Problema de Asignación Multicriterio, Problema de Asignación Fraccionado, Problema de Asignación con Restricciones de Equipo, Problema de Asignación Cuadrático (QAP), Problema de Asignación Robusto y el Problema de Asignación Generalizado (GAP). Las variantes GAP y AP con restricción de equipos son de especial interés en esta investigación.

2.2.1.1 PROBLEMA DE ASIGNACIÓN CON CALIFICACIÓN DE AGENTES

Fue introducido en inicio por Caron et al (1999). En esta variante, no todos los agentes están calificados para procesar todas las tareas, y el objetivo es minimizar el costo total de asignación. Su modelo matemático es:

$$Maximizar\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} q_{ij}X_{ij} \leq 1;\ \forall\, j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} q_{ij}X_{ij} \leq 1;\ \forall\, i = 1,2,\dots,m$$

$$X_{ij} = 0 \text{ ó } 1;\ \forall\, i,j$$

$$[15]$$

Como puede verse, para este caso en general, se está interesado en maximizar la función objetivo. El parámetro q_{ij} es 1 si el agente i está calificado para atender la tarea j y 0 si no lo está. El significado de cada restricción es el mismo que para el caso del AP. Claramente, si $q_{ij} = 0$ entonces $C_{ij} = 0$. Para el caso de minimización, el planteamiento de las restricciones dependerá de la relación entre m y n.

2.2.1.2 PROBLEMA DE ASIGNACIÓN CON CARDINALIDAD

Fue propuesto por Dell'Amico & Martello (1997), y en este se considera que la suma de tareas y agentes asignados tiene un límite. El modelo matemático para esta variante es el siguiente:

$$Minimizar\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} \leq 1;\ \forall\, j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} X_{ij} \leq 1;\ \forall\, i = 1,2,\dots,m$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij} = k$$

$$X_{ij} = 0 \text{ ó } 1;\ \forall\, i,j$$

[16]

Así, a diferencia del AP clásico, no necesariamente todos los agentes o todas las tareas deben estar en una asignación, pero el total de asignaciones es limitado. Los autores proponen como aplicación principal el caso donde un conjunto de operarios deben ser asignados a un conjunto de máquinas, pero solo cierta parte de los operarios y las máquinas pueden ser asignadas.

2.2.1.3 PROBLEMA DE ASIGNACIÓN CUELLO DE BOTELLA

El problema de asignación cuello de botella (BAP) es una variante en la que el objetivo es minimizar el máximo costo de asignación. Las primeras discusiones conocidas sobre esta variante se deben a Ford & Fulkerson (1966). La función objetivo viene dada por:

$$Minimizar\ Z = \max\{C_{ij}X_{ij}\} \qquad [17]$$

Las restricciones en esencia son las mismas del AP, modificándolas dependiendo de la relación entre m y n.

Como ejemplo de aplicación puede citarse el caso de asignación de fondos en diversas inversiones, donde el objetivo es reducir lo más posible el máximo riesgo de inversión.

2.2.1.4 PROBLEMA DE ASIGNACIÓN BALANCEADO

Fue introducido por Martello et al. (1984). En esta variante, el objetivo es minimizar la distancia entre el máximo y el mínimo costo de asignación; es decir, la función objetivo es de la forma:

$$Minimizar\ Z = \max\{C_{ij}X_{ij}\} - \min\{C_{ij}X_{ij}\} \qquad [18]$$

El tratamiento de las restricciones es idéntico a la variante anteriormente expuesta.
En su trabajo, los autores citan como ejemplo de aplicación la selección de suministros para componentes, de modo que la diferencia entre el máximo y el mínimo tiempo de falla esperado sea lo más pequeña posible, permitiendo que las reparaciones de los componentes se den en tiempos similares.

2.2.1.5 PROBLEMA DE ASIGNACIÓN CON MÍNIMA DESVIACIÓN

Fue discutido en inicio por Gupta & Punnen (1988) y por Duin & Volgenant (1991). El objetivo de esta variante es minimizar la diferencia entre el máximo costo y el costo promedio de asignación:

$$Minimizar\ Z = \max\{C_{ij}X_{ij}\} - \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}}{mn} \qquad [19]$$

Como ejemplo puede citarse un Job Shop, en donde el objetivo sea minimizar el tiempo ocioso de las máquinas.

2.2.1.6 PROBLEMA LEXICOGRÁFICO DE CUELLO DE BOTELLA

Es un caso más general del BAP. La idea es encontrar una asignación que no solo minimice el costo de asignación mas grande, sino que a la vez minimice el segundo costo más grande, el tercero, y así sucesivamente. Esta variante fue discutida por Burkard & Rendl (1991) y por Sokkalingam & Aneja (1998). En términos matemáticos, la función objetivo viene dada por:

$$Minimizar \max_1 \{C_{ij}X_{ij}\} \preccurlyeq \max_2 \{C_{ij}X_{ij}\} \preccurlyeq \cdots \max_n \{C_{ij}X_{ij}\} \qquad [20]$$

La función objetivo indica que en primer lugar se minimiza el costo de asignación mas grande, luego el segundo, y así sucesivamente.

2.2.1.7 PROBLEMA DE ASIGNACIÓN K SIGMA

Es un caso más general de los Problema de Asignación Clásico y de Cardinalidad. En esta variante, no hay restricción sobre la cantidad de asignaciones de tareas y agentes. El objetivo consiste en minimizar la suma de cierta cantidad de asignaciones. Grygiel (1981) propone un algoritmo para la solución de este problema. La función objetivo es de la forma:

$$Minimizar\ Z = \max\{\sum_k C_{ij}X_{ij}\} \qquad [21]$$

La letra k indica que solo debe contemplarse esa cantidad de elementos en la función objetivo. Así, el BAP es un caso especial de esta variante, en donde $k = 1$; del mismo modo, el AP clásico es otro caso especial con $k = n$.

2.2.1.8 PROBLEMA DE SEMI ASIGNACIÓN

Es una versión en la que algunas tareas o agentes no son únicos. Se tienen n grupos de tareas, con $m < n$, y cada grupo tiene d_j tareas idénticas, siendo $\sum_j d_j = m$. Su modelo matemático tiene la forma:

$$Minimizar\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} = d_j;\ \forall\ j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} X_{ij} = 1;\ \forall\ i = 1,2,\dots,m$$

$$X_{ij} = 0 \text{ ó } 1;\ \forall\ i,j$$

[22]

Volgenant (1996) aprovecha el hecho de que en la matriz de costos aparecen filas o columnas idénticas, de modo que el proceso de búsqueda del óptimo se acelera.

2.2.1.9 PROBLEMA DE ASIGNACIÓN CATEGORIZADO

Para esta variante se contempla la posibilidad de dependencia en las secuencias de asignación, es decir, no todas las tareas pueden ser ejecutadas de manera simultánea. Las tareas pueden ser categorizadas en grupos, en los que las tareas contenidas tienen relaciones de dependencia entre sí. Para algunas otras variantes del AP, Punnen & Aneja (1993) extienden dichos problemas al caso de categorización y ofrecen detalles sobre las diversas versiones del problema.

2.2.1.10 PROBLEMA DE ASIGNACIÓN MULTICRITERIO

El Problema de Asignación Multicriterio considera más de un objetivo a la vez. Varios modelos estándar han sido establecidos. Geetha & Nair (1993) consideraron dos objetivos de interés, a saber, el costo total de asignación y el makespan de las tareas; Así, en este modelo se incorpora un nuevo parámetro, denominado t_{ij}, siendo este el tiempo que tarda el agente i en procesar la tarea j. Los autores transforman estos dos objetivos en uno solo, mediante la función:

$$Minimizar\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij} + F * \max\{t_{ij}X_{ij}\} \qquad [23]$$

Donde F representa un costo fijo de procesamiento por unidad de tiempo. Las restricciones de esta variante son las mismas del AP. Dada la estructura de la función objetivo, en este modelo se asume que las tareas son procesadas de manera simultánea.

2.2.1.11 PROBLEMA DE ASIGNACIÓN FRACCIONADO

Esta variante presenta como función objetivo una función no lineal en términos de las variables de asignación:

$$Minimizar\ Z = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}X_{ij}} \qquad [24]$$

Las restricciones son las mismas del AP clásico. Shigeno et al. (1995) presenta un método de solución para esta variante.

Como ejemplo de aplicación de este modelo puede verse el intento por minimizar el costo por unidad de tiempo en una asignación de efecto continuo, donde d_{ij} indicaría el tiempo que cada agente tarda en procesar cada tarea.

2.2.1.12 PROBLEMA DE ASIGNACIÓN CON RESTRICCIONES DE EQUIPO

Esta variante se acopla a problemas de la vida real donde existan limitaciones de tiempo, o restricción en la capacidad de los agentes (grado de entrenamiento o rango del personal). El Problema de Calificación de Agentes es un ejemplo de esta variante, pues se impone una restricción que impide que un agente procese cualquier tarea. Foulds & Wilson (1999) fueron los primeros en introducir esta variante del problema de asignación.

2.2.1.13 PROBLEMA DE ASIGNACIÓN CUADRÁTICO

El Problema de Asignación Cuadrático está asociado con la localización de fabricas en nodos geográficos. Su modelo matemático tiene la forma:

$$Minimizar\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{n} C_{ijpq}X_{ij}X_{pq}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} \leq 1;\ \forall\, j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} X_{ij} = 1;\ \forall\, i = 1,2,\dots,m$$

$$X_{ij} = 0 \text{ ó } 1;\ \forall\, i,j$$

[25]

El costo C_{ijpq} en este modelo dependerá de la interacción entre asignaciones. Un ejemplo de dicho costo puede ser la distancia entre dos ubicaciones.
Hillier and Connors (1966) indican como determinar C_{ijpq} dependiendo de la naturaleza del problema de localización. Un estudio muy detallado de aplicaciones del QAP es hecho por Burkard (1984). Dada la complejidad de esta variante del AP, suelen utilizarse heurísticas y metaheurísticas para su resolución.

2.2.1.14 PROBLEMA DE ASIGNACIÓN ROBUSTO

Ésta es una variante propuesta por Kouvelis and Yu (1997) usada para resolver problemas de asignación bajo condiciones de incertidumbre; es decir, problemas de asignación en los que se conocen los posibles escenarios pero no sus probabilidades. Los autores proponen tres versiones distintas para el problema, dando para cada una un método distinto de solución.

2.2.1.15 EL PROBLEMA DE ASIGNACIÓN GENERALIZADO

Ross & Soland propusieron una variante más general del AP, denominada *Problema de Asignación Generalizado* (GAP). En ésta, un agente puede atender cualquier cantidad de tareas. Sin embargo, se tiene capacidad limitada, y cada tarea tiene una tasa de consumo determinada. Se tienen m agentes y n tareas, donde C_{ij} representa el costo de asignar la tarea j al agente i. Formalmente, el GAP se define de manera matemática como sigue:

$$Minimizar\ Z = \sum_{i1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} = 1;\ \forall\, j = 1,2,\dots,n$$

$$\sum_{j=1}^{n} a_{ij} X_{ij} \leq b_i;\ \forall\ i = 1,2,\dots,m$$

$$X_{ij} = 0 \text{ ó } 1;\ \forall\ i,j$$

$$[26]$$

Al igual que en el AP, X_{ij} es la variable de asignación (1 si ésta se realiza y 0 de otro modo). La primera ecuación representa el costo total de asignación. La segunda restricción garantiza que cada tarea sea atendida por un solo agente, mientras que la tercera restricción indica que la capacidad de ningún agente debe ser excedida, siendo b_i la capacidad del agente y a_{ij} la tasa de consumo de recursos por parte de cada tarea.

Fisher & Jaikumar (1981) probaron que este problema es NP-Duro.

Varias han sido las herramientas propuestas para resolver el GAP. Un estudio muy detallado las técnicas más importantes para resolver el GAP es hecho por Cattrysse (1992). De las metodologías más recientes, puede destacarse el trabajo de Amini & Racer (1995), quienes proponen una heurística híbrida para resolver el problema, obteniendo buenos resultados en cuanto a calidad de respuesta y tiempo de cómputo. Özbakir, Baykasoğlu, & Tapkan (2010), proponen una metaheurística basada en el comportamiento natural de las abejas. Entre otros trabajos desarrollados se tienen los de Lorena & Narciso (1996), Narciso & Lorena (1999), Romeijn & Morales (2000), Fern (2001), Cohen, Katzir, & Raz (2006), Jeet & Kutanoglu (2007), Woodcock & Wilson (2010) y Litvinchev, Mata, Rangel, & Saucedo (2010). Zhang y Ong (2007) estudian en su trabajo el GAP multiobjetivo, mientras que Subtil et al. (2010), generalizan al caso de más de dos objetivos. Estos autores exponen condiciones generales para detectar soluciones no dominadas en el espacio de búsqueda, partiendo de la relajación lineal del problema. Sin embargo, dada la naturaleza del problema, el posterior acotamiento de las soluciones crece de manera exponencial con el tamaño del mismo.

3 EL PROBLEMA DE ASIGNACIÓN GENERALIZADO DINÁMICO

En muchos de los problemas de la vida real, el efecto de una asignación no es plano o estático; es decir, el resultado de la asignación marcará el comportamiento del sistema a lo largo de cierto horizonte de tiempo. Ejemplo de ello es la asignación de clientes a centros de distribución en una cadena de suministros, ya que, a lo largo de todo un año, por ejemplo, el cliente será atendido por ese mismo centro. Puede verse que para este tipo de casos los requerimientos de las tareas (tasas de consumo en un GAP) cambian a lo largo del tiempo, al igual que lo hará la capacidad de los agentes, quienes dependen a su vez de un agente principal. Además, en muchos casos se necesita que se procese un mínimo de tareas a lo largo de todo el horizonte de planeación, lo que trae consigo el concepto de nivel de

servicio. Todos los conceptos anteriormente señalados dan lugar a una nueva variante del GAP, denominada *Problema de Asignación Generalizado Dinámico (DGAP)*. Formalmente, el DGAP se define de manera matemática como sigue:

$$Min\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} = 1; \quad \forall\, j$$

$$y_{ijt} \leq X_{ij}; \quad \forall\, i,j,t$$

$$\sum_{j=1}^{n} y_{ijt}r_{ijt} \leq Cap_{it}; \ \forall\, i,t;\ i < m$$

$$Cap_{it} = Cap_{it-1} - \sum_{j=1}^{n} y_{ijt-1}r_{ijt-1} + W_{i(t-k_i)}; \quad \forall\, i\,, t; i < m$$

$$Cap_{it} \leq MaxCap_i; \quad \forall\, i,t; i < m$$

$$\sum_{i=1}^{m-1} W_{i(t-k_i)} + \sum_{j=1}^{n} y_{mjt}r_{mjt} \leq MaxCap_m; \quad \forall\, t$$

$$\sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - y_{ijt}) \leq U_\omega;$$

$$X_{ij}, y_{ijt} = 0 \text{ ó } 1;\ W_{it} \geq 0; \ \forall\, i,j,t$$

[27]

Donde:

$i = 1,2,\dots,m$: $Indice\ para\ las\ agentes, siendo\ m\ el\ agente\ principal.$

$j = 1,2,\dots,n$: $Indice\ para\ los\ tareas.$

$t = 1,2,\dots,T$: $Instante\ de\ tiempo$

X_{ij}: $Variable\ binaria\ de\ asignación\ de\ la\ tarea\ j\ al\ agente\ i.$

y_{ijt}: $Variable\ binaria\ de\ atención\ de\ la\ tarea\ j\ por\ parte\ del\ agente\ i\ en\ el\ tiempo\ t.$

W_{it}: $Recurso\ a\ enviar\ desde\ el\ agente\ principal\ \ hasta\ el\ agente\ secundario\ i\ en\ t.$

Cap_{it}: $Capacidad\ del\ agente\ i\ en\ el\ período\ t.$

r_{ijt}: $Demanda\ de\ recurso\ para\ tarea\ j\ relativa\ al\ agente\ i\ en\ el\ período\ t.$

$MaxCap_i$: $Máxima\ capacidad\ que\ puede\ tener\ el\ agente\ i.$

U_ω: $Máxima\ cantidad\ de\ requerimientos\ ponderados\ no\ atendidos\ permisible.$

k_i: $Tiempo\ necesario\ para\ el\ agente\ secundario\ i\ disponga\ de\ la\ capacidad\ dada\ por\ m.$

ω_j: $Ponderación\ de\ la\ importancia\ relativa\ de\ la\ tarea\ j.$

La primera restricción asegura que una tarea sea asignada a un único agente. La segunda restricción garantiza que en cualquier instante de tiempo una tarea sea procesada solo por el agente al cual fue asignada. La tercera restricción indica que en ningún instante de tiempo la capacidad de un agente secundario debe ser excedida. La cuarta restricción permite calcular la capacidad de cualquier agente secundario. La quinta restricción indica que cada agente secundario tiene un tope de capacidad, mientras que la sexta garantiza que la capacidad del agente principal nunca sea excedida. La última restricción establece que la cantidad de tareas ponderadas no atendidas a lo largo de todo el horizonte no puede exceder un tope preestablecido.

Dado que en la vida real no necesariamente todas las tareas tienen la misma importancia, la atención o procesamiento de cada una tendrá una importancia relativa con respecto a las demás (ω_j). La constante U_ω es tal que $U_\omega \leq T \sum_{i,j/r_{jt}>0} \omega_j X_{ij}$, ya que, de lo contrario, la última restricción no tendría sentido real (siempre sería satisfecha).

En este modelo se asume que los requerimientos no son cautivos; es decir, que si una tarea no es atendida en un determinado instante de tiempo, no podrá ser procesada en otro instante (Por ejemplo, un cliente preferiría comprar a otra empresa). De igual modo, se asume que las tareas pueden ser procesadas de manera instantánea por parte de los agentes.

3.1 COMPLEJIDAD DEL DGAP

A simple vista, el DGAP parece ser un modelo extremadamente complejo, pues el número de variables crece rápidamente a medida que se incrementa el tamaño del problema. El total de variables para este modelo es:

- $Variables\ \ binarias\ de\ asignación: mn$
- $Variables\ binarias\ de\ atención: mnT$
- $Variables\ continuas\ de\ envíos\ a\ agentes\ secundarios: mT$
- $Variables\ continuas\ de\ capacidad: (m-1)T$

Luego, el total de variables binarias para un modelo del DGAP es de:

$$Total\ variables\ binarias = mn + mnT = mn(T+1) \quad [28]$$

Así, un modelo con solo 10 agentes, 10 tareas y 10 instantes de tiempo tendrá un total de 1100 variables binarias, con lo que explorar todas las posibles soluciones (2^{1100}) sería algo imposible. Sin embargo, este modelo tiene propiedades matemáticas importantes, que serán descritas más adelante.

Debe justificarse el porqué usar una Metaheurística para tratar el DGAP (En este caso, Algoritmos Genéticos). Dicha justificación consiste en mostrar que el DGAP es NP-Duro.

- ***Teorema 1:*** *El DGAP es NP-Duro*

Prueba: Basta con demostrar que el DGAP reduce al GAP. Este último puede ser reescrito como:

$$Minimizar\ Z = \sum_{i1}^{m}\sum_{j=1}^{n} C_{ij}X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} = 1;\ \forall\ j = 1,2,\dots,n$$

$$y_{ij} \leq X_{ij};\ \forall\ i,j$$

$$\sum_{j=1}^{n} a_{ij} y_{ij} \leq b_i;\ \forall\ i = 1,2,\dots,m$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} (X_{ij} - y_{ij}) \leq 0$$

$$y_{ij}, X_{ij} = 0 \text{ ó } 1;\ \forall\ i,j$$

Si en el DGAP se hace $T = 1$, se obtiene:

$$Min\ Z = \sum_{i=1}^{m}\sum_{j=1}^{n} C_{ij} X_{ij}$$

$$s.a:$$

$$\sum_{i=1}^{m} X_{ij} = 1;\quad \forall\ j$$

$$y_{ij} \leq X_{ij};\quad \forall\ i,j$$

$$\sum_{j=1}^{n} y_{ij} r_{ij} \leq Cap_i;\ \ \forall\ i < m$$

$$Cap_{i0} = Cap_{i0}\ ;\ \ \forall\ i\ < m$$

$$Cap_i \leq MaxCap_i;\quad \forall\ i,t; i < m$$

$$\sum_{j=1}^{n} y_{mj} r_{mj} \leq MaxCap_m;$$

$$\sum_{(i,j)/r_{ij}>0;} \omega_j (X_{ij} - y_{ij}) \leq U_\omega;$$

$$X_{ij}, y_{ij} = 0 \text{ ó } 1;$$

La cuarta restricción es trivial, ya que cualquier valor es igual a sí mismo. De la tercera y la quinta, si $\sum_{j=1}^{n} y_{ij} r_{ij} \leq Cap_i$ y $Cap_i \leq MaxCap_i$, entonces $\sum_{j=1}^{n} y_{ij} r_{ij} \leq MaxCap_i\ \forall\ i < m$. Como $\sum_{j=1}^{n} y_{mj} r_{ij} \leq MaxCap_m$, entonces se tiene que $\sum_{j=1}^{n} y_{ij} r_{ij} \leq MaxCap_i\ \forall\ i$. Luego, tomando $a_{ij} = r_{ij}, \omega_j = 1,\ MaxCap_i = b_i, \forall\ i,j$ y

$U_\omega = \sum_{i,j/r_{ij}>0}(1)(X_{ij} - y_{ij}) = 0$, se deduce el modelo transformado del GAP. Entonces, como el DGAP reduce en tiempo polinomial al GAP y dado que este último es NP-Duro, se concluye que el DGAP también es NP-Duro.

3.2 FUNDAMENTACIÓN TEÓRICA DEL DGAP

- ***Definición 1:*** Sea un modelo matemático de la forma:

$$\min Z = f(X)$$

$$s.a:$$

$$g_q(X,Y) \leq b_q;\ \forall\ q = 1,2,\dots,Q$$

$$X \in S_X \subseteq R^n, Y \in S_Y \subseteq R^n;\ b_q \in R\ \forall\ q;\ g_q(X,Y): R^n \to R\ \forall\ q$$

[29]

Donde S_X y S_Y representan los conjuntos de posibles valores de X y Y, respectivamente. Se dice que el modelo es de O*rden Parcialmente Polinomial* si, $\forall\ X \in S_X$, se verifica y calcula (en caso de que exista) un Y factible en S_Y en tiempo polinomial. Esto es, dado cualquier $X = X_0$ en S_X, existe un algoritmo que determina si existe o no un Y factible (calculándolo en caso de que exista) empleando un tiempo acotado por un polinomio.

- ***Definición 2:*** Sea el modelo descrito bajo las condiciones de la Definición 1. Se dice que el modelo es de O*rden Parcialmente Polinomial al nivel* $1-\gamma$ si, dada una solución factible en $X = X_0$, existe un algoritmo que el $(1-\gamma) * 100\%$ de las veces calcula un Y factible en tiempo polinomial.

- ***Definición 3:*** Sea $X = X_0 \in S_X$ en [29]. Suponga el modelo:

$$\max Z' = b_{q'} - g_{q'}(X_0, Y)$$

$$s.a:$$

$$g_q(X_0, Y) \leq b_q;\ \forall\ q \neq q'$$

$$X \in S_X \subseteq R^n, Y \in S_Y \subseteq R^n;\ b_q \in R\ \forall\ q;\ g_q(X,Y): R^n \to R$$

[30]

Se dice entonces que q' es una *restricción de máximo factible* si q' es factible en todo $Y \in S_Y$ que maximice Z' .

La anterior definición implica que, si existe una restricción q' de máximo factible en [30], entonces existe un $Y \in S_Y$ factible para [29].

- ***Definición 4: Sea*** una función $f(X)$, con $X \in S; S \neq \emptyset$. Se dice que f está acotada superiormente en S si existe $X^* \in S$ tal que $f(X^*) \geq f(X), \forall\ X \in S$.

- ***Definición 5: Sea*** una función $f(X)$, con $X \in S; S \neq \emptyset$. Se dice que f está acotada inferiormente en S si existe $X^* \in S$ tal que $f(X^*) \leq f(X), \forall\ X \in S$.

- ***Teorema 2:*** *Sea q' una restricción de [29], en donde $X = X_0$. Sea además $Y = Y^* \in S_Y$ un valor que maximiza Z' en [30]. Si $Y = Y^*$ no satisface q', entonces no existe ningún valor $Y \in S_Y$ para el que la pareja X_0, Y sea factible.*

Prueba: Se sabe que $b_{q'}$ es una constante. Al maximizar $b_{q'} - g_{q'}(X_0, Y)$ se minimiza $g_{q'}(X_0, Y)$. Por hipótesis, $Y = Y^*$ *es tal que:*

$$g_{q'}(X_0, Y^*) \leq g_{q'}(X_0, Y)\ \ \forall\ Y \in\ S_Y$$

$$g_{q'}(X_0, Y^*) - b_{q'} \leq g_{q'}(X_0, Y) - b_{q'}\ \ \forall\ Y \in\ S_Y \quad [31]$$

Si $Y = Y^*$ no satisface q', entonces:

$$g_{q'}(X_0, Y^*) > b_{q'} \to 0 < b_{q'} - g_{q'}(X_0, Y^*) \qquad [32]$$

De [31] y [32] se tiene:

$$0 < b_{q'} - g_{q'}(X_0, Y^*) \leq g_{q'}(X_0, Y) - b_{q'}\ \ \forall\ Y \in\ S_Y$$

$$0 < g_{q'}(X_0, Y) - b_{q'}\ \ \forall\ Y \in\ S_Y$$

$$g_{q'}(X_0, Y) > b_{q'}\ \ \forall\ Y \in\ S_Y$$

Así, el último resultado muestra que ningún otro Y genera una solución factible en q', y por ende, tampoco lo hará para el modelo completo. La prueba queda completa.

- ***Teorema 3:*** *Si [29] tiene una solución factible en $X = X_0$ para algún $Y \in S_Y$, y Z' está acotada superiormente $\forall\, X, Y$, entonces existe $Y^* \in S_Y$ que maximiza Z' en [30].*

Prueba: En primer lugar supongamos que [29] solo tiene una solución factible, dada por el punto X_0, Y_0. Al ser factible este modelo, [30] también lo será, y como solo se tiene un punto factible, $Z'_{max} = b_{q'} - g_{q'}(X_0, Y_0)$, de donde $Y^* = Y_0$. Suponer ahora que existe más de un punto factible. Sea $S'_Y \subseteq S_Y$ el conjunto de todos los valores factibles para Y en [29], que también son valores factibles para [30]. Por hipótesis, Z' está acotada superiormente para cualquier pareja X, Y que esté en un conjunto no vacío. Luego, al ser $X = X_0$ y $S'_Y \neq \emptyset$ (Por hipótesis de factibilidad), existe $Y^* \in S'_Y$ para el que $b_{q'} - g_{q'}(X_0, Y^*) \geq b_{q'} - g_{q'}(X_0, Y)$; $\forall\, Y \in S'_Y$, con lo que la prueba queda completa.

Del anterior teorema se desprende el siguiente corolario:

- ***Corolario 1:*** *Si [29] tiene una solución factible en $X = X_0$ para algún $Y \in S_Y$, y Z' está acotada superiormente $\forall\, X, Y$, entonces q' en [30] es una restricción de máximo factible.*

Prueba: Del teorema anterior, existe Y^* factible que maximiza Z'. Además, al ser Y^* factible en [29], q' es también factible, razón por la que es una restricción de máximo factible.

- ***Teorema 4: Sea*** *q' como se describe en [30] y $Y = Y^* \in S_Y$ un valor que maximiza $b_{q'} - g_{q'}(X_0, Y)$. Si existe un algoritmo que en un tiempo polinomial*

verifique la existencia de Y^*, calculándolo, en caso de que exista, entonces [29] es un modelo de orden parcialmente polinomial.

Prueba: En caso de que se encuentre Y^* y éste no satisfaga q', del Teorema 2, ningún otro Y en S_Y satisface q' y mucho menos el resto de restricciones, por lo que se verificaría la inexistencia de un Y factible en S_Y para [29] en tiempo polinómico. En caso de que Y^* satisfaga q', por definición de restricción de máximo factible, la solución X_0, Y es factible, siendo $Y = Y^* \in S_Y{}'$ calculado en tiempo polinomial. Por tanto, la verificación de la existencia de un $Y^* \in S_Y{}'$ factible y su cálculo se hacen siempre en tiempo polinómico, razón por la cual el modelo sería de orden parcialmente polinomial.

De los teoremas 2 y 3 y del corolario 1, se tiene que, dado un $X = X_0$, para verificar si existe algún Y que haga factible [29], basta con resolver [30] y calcular (si existe) Y^*. Del teorema 4, si lo anterior puede hacerse en tiempo polinomial, entonces el modelo dado por [29] resulta ser parcialmente polinomial. Estas propiedades matemáticas son extremadamente importantes para el diseño del algoritmo propuesto para resolver el DGAP.

- ***Teorema 5:*** *Sea el DGAP definido en [27]. Entonces, la función $Z' = U_\omega - \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - y_{ijt})$ es superiormente acotada.*

Prueba: De la definición del DGAP, se tiene que $X_{ij} \geq y_{ijt} \ \forall\, i, j, t$. Así:

$$\sum_{i,j,t/r_{ijt}>0} \omega_j(X_{ij} - y_{ijt}) \geq 0; \rightarrow - \sum_{i,j,t/r_{ijt}>0} \omega_j(X_{ij} - y_{ijt}) \leq 0;$$

$$\rightarrow U_\omega - \sum_{i,j,t/r_{ijt}>0} \omega_j(X_{ij} - y_{ijt}) = Z' \leq U_\omega$$

Luego, del último resultado se observa fácilmente que Z' alcanza su máximo valor en $X_{ij} = y_{ijt}, \forall\, i, j, t$, razón por la cual Z' está acotada superiormente.

- ***Teorema 6:*** *Sea el DGAP definido en [27]. Sean* $X = \bigcup_{i,j} X_{ij}$ *y* $Y = \bigcup_{j,t} y_{ijt}, f_{ijt}, W_{it}$. *Si el DGAP tiene solución factible cuando* $X = X_0$, *entonces,* $q' : \sum_{i,j,t/r_{ijt}>0} \omega_j (X_{ij} - y_{ijt}) \leq U_\omega$ *es una restricción de máximo factible.*

Prueba: El DGAP tiene la misma estructura del modelo teórico expuesto en [29] ,ya que, solo X interviene en la función objetivo. Sea el modelo:

$$max\ Z' = U_\omega - \sum_{i,j,t/r_{ijt}>0} \omega_j (X_{ij} - y_{ijt})$$

$$s.a:$$

$$y_{ijt} \leq X_{ij};\quad \forall\, i,j,t$$

$$\sum_{j=1}^{n} y_{ijt} r_{ijt} \leq Cap_{it};\ \ \forall\, i,t;\ i < m$$

$$Cap_{it} = Cap_{it-1} - \sum_{j=1}^{n} y_{ijt-1} r_{ijt-1} + W_{i(t-k_i)};\quad \forall\, i\ , t; i < m$$

$$Cap_{it} \leq MaxCap_i;\quad \forall\, i,t; i < m$$

$$\sum_{i=1}^{m-1} W_{i(t-k_i)} + \sum_{j=1}^{n} y_{mjt} r_{ijt} \leq MaxCap_m;\quad \forall\, t$$

$$X_{ij}, y_{ijt} = 0 \text{ ó } 1;\ W_{i,t} \geq 0;\ \ \forall\, i,j,t$$

[33]

En el anterior modelo, se obvia la primera restricción ya que se asume por hipótesis que es factible. El teorema 5 garantiza que Z' es acotada superiormente en cualquier espacio de soluciones factible, y dada la hipótesis de existencia de factibilidad, según el teorema 3, existe Y^* que maximiza Z' , por lo que, a partir del corolario 1, se concluye que q' es una restricción de máximo factible.

El siguiente teorema no es más que la demostración de la cota superior que George Dantzig introdujo para el Problema de la Mochila, el cual, no es discutido en este trabajo.

- ***Teorema 7:*** *Sea el modelo:*

$$max\ Z'' = \sum_{i=1}^{n} \omega_i x_i$$

$$s.a:$$

$$\sum_{i=1}^{n} r_i x_i \leq R$$

$$0 \leq x_i \leq 1\ \forall\ i$$

$$[34]$$

Donde $\omega_i, r_i > 0\ \forall\ i$. *Suponga que* $\frac{\omega_1}{r_1} \geq \frac{\omega_2}{r_2} \geq \cdots \geq \frac{\omega_n}{r_n}$. *Sea además* $k \leq n$ *tal que* $\sum_{i=1}^{k} r_i < R$ *y* $\sum_{i=1}^{k+1} r_i \geq R$. *Entonces, si* $x'_1 = x'_2 = \cdots = x'_k = 1, x'_{k+1} = \frac{R - \sum_{i=1}^{k} r_i}{r_{k+1}}$, *y* $x'_i = 0\ \forall\ i > k+1,\quad Z''(x'_1, x'_2, \ldots, x'_n) \geq Z''(x_1, x_2, \ldots, x_n)$ *para todo* $x_1, x_2, \ldots, x_n$ *que satisfaga [34].*

Prueba: De la hipótesis se tiene que:

$$\sum_{i=1}^{n} x'_i r_i = \sum_{i=1}^{k} (1) r_i + (r_{k+1}) \frac{R - \sum_{i=1}^{k} r_i}{r_{k+1}} + \sum_{i=k+2}^{n} (0) r_i = R$$

Si $x_1, x_2, \ldots, x_n$ es factible entonces:

$$\sum_{i=1}^{n} x_i r_i \leq R = \sum_{i=1}^{n} x'_i r_i \rightarrow \sum_{i=1}^{n} (x'_i - x_i) r_i \geq 0$$

Sean $Z_1 = Z''(x'_1, x'_2, \ldots, x'_n)$ y $Z_2 = Z''(x_1, x_2, \ldots, x_n)$. Ahora, $\forall\ i \leq k$ se tiene que $(x'_i - x_i) \geq 0$ y $\frac{\omega_i}{r_i} \geq \frac{\omega_{k+1}}{r_{k+1}}$, por lo que $(x'_i - x_i) \frac{\omega_i}{r_i} \geq (x'_i - x_i) \frac{\omega_{k+1}}{r_{k+1}}$. Por tanto:

$$Z_1 - Z_2 = \sum_{i=1, i \neq k+1}^{n} (x'_i - x_i)\, r_i + (x'_{k+1} - x_{k+1}) r_{k+1}$$

$$= \sum_{i=1, i \neq k+1}^{n} (x'_i - x_i) \frac{\omega_i}{r_i} r_i + (x'_{k+1} - x_{k+1}) \frac{\omega_{k+1}}{r_{k+1}} r_{k+1}$$

$$\geq \sum_{i=1, i\neq k+1}^{n} (x'_i - x_i)\frac{\omega_{k+1}}{r_{k+1}} r_i + (x'_{k+1} - x_{k+1})\frac{\omega_{k+1}}{r_{k+1}} r_{k+1}$$

$$= \sum_{i=1}^{n} (x'_i - x_i)\frac{\omega_{k+1}}{r_{k+1}} r_i = \frac{\omega_{k+1}}{r_{k+1}} \sum_{i=1}^{n} (x'_i - x_i)\, r_i \geq 0$$

Luego, la prueba queda completa.

A la regla que permite calcular el óptimo de [34] se le denomina *regla voraz,* y consiste en hacer asignaciones ordenando los elementos $\frac{\omega_i}{r_i}$ de mayor a menor, hasta que se asigne toda la capacidad disponible. Dicho teorema permite establecer una cota superior para el caso en el que los x_i sean binarios.

- ***Teorema 8:*** *Si* X_{ij} *es conocido* $\forall\, i, j$, *entonces maximizar* $U_\omega - \sum_{i,j,t/r_{ijt}>0} \omega_j (X_{ij} - y_{ijt})$ *equivale a maximizar* $\sum_{i,j,t/r_{ijt}>0} \omega_j y_{ijt}$.

Prueba: Sean y^*_{ijt} los puntos que definen el máximo de la primera función. Así:

$$U_\omega - \sum_{i,j,t/r_{ijt}>0} \omega_j (X_{ij} - y^*_{ijt}) \geq U_\omega - \sum_{i,j,t/r_{ijt}>0} \omega_j (X_{ij} - y_{ijt})\,;\ \forall\ y_{ijt}$$

$$\rightarrow - \sum_{i,j,t/r_{ijt}>0} \omega_j X_{ij} + \sum_{i,j,t/r_{ijt}>0} \omega_j y^*_{ijt} \geq - \sum_{i,j,t/r_{ijt}>0} \omega_j X_{ij} + \sum_{i,j,t/r_{ijt}>0} \omega_j y_{ijt}\,;\ \forall\ y_{ijt}$$

$$\rightarrow \sum_{i,j,t/r_{ijt}>0} \omega_j y^*_{ijt} \geq \sum_{i,j,t/r_{ijt}>0} \omega_j y_{ijt}\,;\ \forall\ y_{ijt}$$

Así, la prueba queda completa.

- ***Teorema 9:*** *El modelo [33] es equivalente a :*

$$max\ Z' = U'_{\omega} - \sum_{i,j,t'/r_{ijt'}>0} \omega_j (X_{ij} - y_{ijt'})$$

$$s.a:$$

$$y_{ijt'} \leq X_{ij};\quad \forall\, i,j,t'$$

$$\sum_{j=1}^{n} y_{ijt'} r_{ijt'} \leq Cap_{it'};\ \ \forall\, i,t';\ i < m$$

$$Cap_{it} = Cap_{it'-1} - \sum_{j=1}^{n} y_{ijt'-1} r_{ijt'-1} + W_{it'};\quad \forall\, i\,, t'; i < m$$

$$Cap_{it'} \leq MaxCap_i;\quad \forall\, i,t'; i < m$$

$$\sum_{i=1}^{m-1} W_{it'} + \sum_{j=1}^{n} y_{mjt'} r_{ijt'} \leq MaxCap_m;\quad \forall\, t'$$

$$X_{ij}, y_{ijt'} = 0 \text{ ó } 1;\ W_{i,t'} \geq 0;\ \ \forall\, i,j,t'$$

[35]

Donde $U'_{\omega} = U_{\omega} - \sum_{(i,j,t)/r_{ijt}>0;\ t\leq k_i} \omega_j (X_{ij} - y_{ijt})$; $1 \leq t' \leq T$ y $r_{ijt'}$ es tal que:

$$r_{ijt'} = \begin{cases} r_{ij(t+k_i)} : X_{ij} = 1\,, t' \leq T - k_i \\ 0\ en\ otro\ caso \end{cases}$$

Prueba: Cualquier agente i puede disponer de recursos del agente principal solo a partir del instante $1 + k_i$. Luego, los requerimientos que compiten por esta capacidad inicial son $r_{ij(1+k_i)}$, siendo i el agente que atiende a la tarea j. Entonces, cualquier requerimiento que se encuentre antes de $1 + k_i$ solo podrá ser atendido con la capacidad inicial (en caso de que exista) del respectivo agente, por lo que en realidad $\sum_{(i,j,t)/r_{ijt}>0;\ t\leq k_i} \omega_j (X_{ij} - y_{ijt})$ debe ser extraído de U_{ω} en la función objetivo. Si t' representa el instante de tiempo en donde los requerimientos compiten directamente por capacidad, el requerimiento de cada tarea en ese instante será $r_{ij(t'+k_i)}$, y la transformación implica que la disposición de $MaxCap_m$ en cualquier instante de tiempo es instantánea. Ahora bien, el máximo tiempo inicial es T, por lo que solo en los instantes relativos $t' \leq T - k_i$ existen requerimientos. Así. la prueba queda completa.

- ***Definición 6:*** Sea el modelo descrito en [35], con X_{ij} conocido $\forall\, i,j$ y $\sum_{i=1}^{m} X_{ij} = 1;\ \ \forall\, j$. Sea además una cantidad $\Delta_{t'} > 0$ tal que $\Delta_{t'} \leq MaxCap_m - \sum_{i=1}^{m-1} W_{it'} - \sum_{j=1}^{n} y_{mjt'} r_{ijt'}$ para algún instante t'. Se dice que $\Delta_{t'}$ es *transferible* si existe $i' < m$ en $t'+1$ tal que $\sum_{j=1}^{n} X_{i'j} r_{ij(t'+1)} + \Delta_{t'} \leq Cap_{i'(t'+1)}$.

- ***Definición 7:*** Sea $\Delta_{t'}$ una cantidad transferible en [35] para algún agente i'. Se dice que $\Delta_{t'}$ es *consumada* si en $t'+1$ $Cap_{i'(t'+1)} = \Delta_{t'} + W_{i'(t'+1)}$

- ***Teorema 10:*** *Sea el modelo expresado en [35]. Si existe una cantidad transferible $\Delta_{t'}$ y $MaxCap_m < \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} r_{ij(t'+1)}$, entonces ésta es consumada en la solución óptima.*

Prueba: El hecho de que una cantidad transferible sea consumada implica que sea usada para alguna asignación en el instante $t'+1$. Como $MaxCap_m$ es la máxima capacidad disponible en cualquier instante de tiempo, dado que las asignaciones dependen de dicha capacidad, resulta trivial que:

$$Z'(MaxCap_m) \leq Z'(MaxCap_m + \Delta_{t'})$$

Por tanto, la prueba queda completa.

- ***Teorema 11:*** *Sea el modelo expresado en [35] (o [33]). Si $\bigcup_{i,j,t} {y^*}_{ijt}$ es el conjunto que maximiza $U_\omega - \sum_{(i,j,t)/r_{ijt}>0;\ \omega_j} (X_{ij} - y_{ijt})$, entonces el modelo [35] solo tendrá soluciones factibles si U_ω es tal que $\sum_{(i,j,t)/r_{ijt}>0;\ \omega_j} (X_{ij} - {y^*}_{ijt}) \leq U_\omega \leq T \sum_{i,j/r_{ijt}>0} \omega_j X_{ij}$, siendo X_{ij} conocido $\forall\, i,j$.*

Prueba: La demostración es trivial, pues basta con hacer que q' en términos de ${y^*}_{ijt}$ sea factible.

- ***Teorema 12:*** *Sea A un algoritmo que obtiene una solución* $\cup_{i,j,t} {y'}_{ijt}$ *a [35]. Si las instancias solo dependen de* U_ω *y* X_{ij} *es conocido* $\forall\, i,j$*. Entonces A obtiene soluciones factibles para [29] el* $\frac{\sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{\sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}} * 100\%$ *de las veces.*

Prueba: Del teorema 10, el máximo rango de soluciones factibles en función de parámetro U_ω es $\sum_{(i,j,t)/r_{ijt}>0;}\ \omega_j(X_{ij} - {y^*}_{ijt}) \le U_\omega \le T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij}$. Por hipótesis:

$$U_\omega - \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - {y'}_{ijt}) \le U_\omega - \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - {y^*}_{ijt})$$

$$\sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - {y^*}_{ijt}) \le \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - {y'}_{ijt})$$

Por factibilidad, U_ω debe ser tal que $U_\omega \ge \sum_{(i,j,t)/r_{ijt}>0;}\ \omega_j(X_{ij} - {y'}_{ijt})$, lo que implica:

$$\sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - {y^*}_{ijt}) \le \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - {y'}_{ijt}) \le U_\omega \le T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij}$$

Como el máximo rango de soluciones factibles para U_ω es $[\sum_{(i,j,t)/r_{ijt}>0;}\ \omega_j(X_{ij} - {y^*}_{ijt}), T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij}]$, y el rango de soluciones factibles para el algoritmo A es $[\sum_{(i,j,t)/r_{ijt}>0;}\ \omega_j(X_{ij} - {y'}_{ijt}), T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij}]$, entonces la proporción de soluciones factibles para A es:

$$\to \frac{T\sum_{i,\frac{j}{r_{ijt}}>0} \omega_j X_{ij} - \sum_{\frac{i,j,t}{r_{ijt}}>0;}\ \omega_j(X_{ij} - {y'}_{ijt})}{T\sum_{i,\frac{j}{r_{ijt}}>0} \omega_j X_{ij} - \sum_{\frac{i,j,t}{r_{ijt}}>0;}\ \omega_j(X_{ij} - {y^*}_{ijt})}$$

$$= \frac{T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij} - \sum_{i,j,t/r_{ijt}>0} \omega_j X_{ij} + \sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij} - \sum_{i,j,t/r_{ijt}>0} \omega_j X_{ij} + \sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}}$$

$$= \frac{T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij} - T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij} + \sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij} - T\sum_{i,j/r_{ijt}>0} \omega_j X_{ij} + \sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}}$$

$$= \frac{\sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{\sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}}$$

- ***Corolario 2:*** *Un algoritmo A obtiene soluciones factibles para [35] el* $\frac{\sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{\sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}} * 100\%$ *de las veces en instancias que solo dependen de* U_ω *y* X_{ij} *es conocido* $\forall\, i,j$ *si y solo si tiene una eficiencia del* $\frac{\sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{\sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}} * 100\%$ *al resolver [35].*

Prueba: La demostración es trivial. El término $\frac{\sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{\sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}}$ representa, según el teorema 12, la proporción de soluciones factibles para [35] que A encuentra cuando las instancias dependen solo de U_ω, manteniendo los otros términos constantes. De igual modo, $\frac{\sum_{i,j,t/r_{ijt}>0} \omega_j {y'}_{ijt}}{\sum_{i,j,t/r_{ijt}>0} \omega_j {y^*}_{ijt}}$ representa la eficiencia de A al resolver [35].

- ***Teorema 13:*** *Sea A un algoritmo para el que se asume una probabilidad* $p \geq 1-\gamma$ *de encontrar un conjunto* $\cup_{i,j,t} {y'}_{ijt}$ *factible para cualquier instancia de [35] (en caso de que exista), en donde* X_{ij} *es conocido* $\forall\, i,j$*. Sean* K *instancias de [35], donde, para cada una, A encuentra el* $\rho_k * 100\%$ *de las soluciones factibles en función de* U_ω*. Entonces, debe cumplirse que* $\sum_{k=0}^{K'} \binom{K}{k} (1-\gamma)^k \gamma^{K-k} > \alpha$*, para que* p *sea estadísticamente verdadera, siendo* $1-\alpha$ *la confiabilidad de admitir* $1-\gamma$ *como verdadera y* K' *el número de instancias para las que* $\rho_k \geq 1-\gamma$*.*

Prueba: Como tal, el teorema conlleva consigo una prueba de hipótesis. Supongamos la prueba:

$$H_0: p = 1-\gamma$$
$$H_1: p < 1-\gamma$$

Ahora, toda prueba de hipótesis se desarrolla asumiendo H_0 como verdadera. Así, puede considerarse un experimento Bernoulli, donde existe un éxito si $\rho_k \geq 1-\gamma$ y un fracaso en el caso contrario. Sea K' el total de éxitos entre las instancias. Si H_0 es falsa, la muestra debe proporcionar evidencia de ello, haciendo poco probable que el total de éxitos sea en realidad K' o menor (según la hipótesis alternativa). Entonces, dado que es necesario calcular la probabilidad de cierta cantidad de éxitos en un experimento Bernoulli, se requiere el uso de la distribución binomial, permitiendo ésta calcular el valor P de la prueba:

$$P\ valor = \sum_{k=0}^{K'} \binom{K}{k} (1-\gamma)^k \gamma^{K-k}$$

Como el valor P es la mínima probabilidad a partir de la cual se rechaza la hipótesis nula, para que p sea estadísticamente verdadero, no debe rechazarse H_0, lo que implica que $P\ valor > \alpha$.

- ***Teorema 14:*** *Sea A un algoritmo para el que se asume una probabilidad* $p \geq 1-\gamma$ *de encontrar un conjunto* $\bigcup_{i,j,t} y'_{ijt}$ *factible para cualquier instancia de [35] (en caso de que exista). Si* $1-\alpha$ *es la confiabilidad de asumir* p *como verdadero y* β *es la probabilidad de no darse cuenta de que la verdadera probabilidad de que A encuentre soluciones factibles para [35] es* $p' = p - \delta$*, con* $0 \leq \delta \leq p$*, entonces* K *y* K' *debe ser tales que:*

$$\sum_{k=0}^{K'-1} \binom{K}{k} (1-\gamma)^k \gamma^{K-k} = \alpha$$

$$\sum_{k=K'}^{K} \binom{K}{k} (1-\gamma-\delta)^k (\gamma+\delta)^{K-k} = \beta$$

Prueba: Suponga que K' es el valor a partir del cual, todo $K < K'$ conlleva a rechazar H_0, así:

$$\alpha = \sum_{k=0}^{K'-1} \binom{K}{k} (1-\gamma)^k \gamma^{K-k}$$

Por definición, β es la probabilidad de no rechazar la hipótesis nula cuando $p' = p - \delta$ es la verdadera probabilidad de éxito. Así:

$$\beta = \sum_{k=K'}^{K} \binom{K}{k} (1-\gamma-\delta)^k (\gamma+\delta)^{K-k}$$

Por tanto, la demostración queda completa.

Los teoremas 2 al 14 y las definiciones que se encuentran entre ellos tienen como objetivo establecer un contexto claro para la elaboración de un algoritmo que permita mostrar que el DGAP es de orden parcialmente polinomial al nivel $1-\gamma$, con un γ lo suficientemente pequeño. Esto se logra, según resultados anteriores, encontrando un algoritmo que en la mayoría de los casos encuentre una solución factible cercana al óptimo de [33], o lo que es lo mismo, según el teorema 9, cercana al óptimo de [35], pues el objetivo está dado por una restricción de máximo factible (teorema 6). Del teorema 9 se deduce que, un algoritmo que resuelva el DGAP sin tiempos de disposición de recursos por parte de los agentes secundarios, también resuelve el DGAP establecido en [35]. Este último resultado es de gran importancia a nivel experimental, ya que la generación de las instancias para la evaluación del algoritmo no requerirá el uso de los k_i. El teorema 8 indica que [33] o [35] son en realidad una generalización del problema de la mochila, pues tienen el mismo objetivo, mientras que el teorema 7 permite establecer una cota superior para este último problema, por lo que parece razonable emplear regla voraz para resolver el problema de interés aquí. Además de la regla voraz, el teorema 10 sugiere hacer uso de las cantidades transferibles en cualquier algoritmo que intente resolver [35]. Los teoremas 11, 12, 13 y 14 constituyen la base teórica para el desarrollo de pruebas experimentales que permitan hacer una estimación de γ.

3.3 FASE DE EXPERIMENTACIÓN 1

En esta primera fase de experimentación se propone un algoritmo para la resolución de diversas instancias del modelo [35]. Luego de esto se hace un estudio estadístico detallado para una estimación razonable de γ, con el ánimo de mostrar que [35] es un modelo de orden parcialmente polinomial de alto nivel (γ muy pequeño).

El algoritmo propuesto para resolver [35] se denomina RS (asumiendo $k_i = 0 \; \forall \; i$ y X_{ij} conocido $\forall \; i, j$, con $\sum_{i=1}^{m} X_{ij} = 1$), y se describe a continuación.

.3.1 ALGORITMO RS

- Paso 1: Calcular los factores $\frac{w_j}{r_{ijt}}$, $\forall \; r_{ijt} > 0, X_{ij} = 1$ y agruparlos en orden decreciente, de modo que $\frac{w^1{}_{j_1}}{r^1{}_{i_1 j_1 t_1}} \geq \frac{w^2{}_{j_2}}{r^2{}_{i_2 j_2 t_2}} \geq \cdots \geq \frac{w^l{}_{j_l}}{r^l{}_{i_l j_l t_l}} \geq \cdots \geq \frac{w^L{}_{j_L}}{r^L{}_{i_L j_L t_L}}$ $\forall \; i_l j_l t_l / \; l \leq L \leq nT; \; j_l \leq n; \; t_l \leq T$. Hacer $Z' = 0, l = 0, \; aux = 0, Cap_{it} = 0, W_{it} = 0, y_{ijt} = 0, \forall \; i, j, t$.

- Paso 2: Hacer $l = l + 1$. Si $l > L$ hacer $j = 0, t = 2, \Delta_t = MaxCap_{m1}$ e ir al paso 4, sino ir al paso 3.

- Paso 3: Si $i_l < m$, $Cap_{i_l t_l} + r_{i_l j_l t_l} \leq MaxCap_{i_l}$ y $MaxCap_{mt_l} \geq r_{i_l j_l t_l}$ entonces hacer $Cap_{i_l t_l} = Cap_{i_l t_l} + r_{i_l j_l t_l}, W_{i_l t_l} = W_{i_l t_l} + r_{j_l t_l}, y_{i_l j_l t_l} = 1, MaxCap_{mt_l} = MaxCap_{mt_l} - r_{i_l j_l t_l}$, y $Z' = Z' + w_{j_l}$. Si $i_l = m$, y $MaxCap_{mt_l} \geq r_{i_l j_l t_l}$ entonces hacer $y_{mj_l t_l} = 1, MaxCap_{mt_l} = MaxCap_{mt_l} - r_{i_l j_l t_l}$ y $Z' = Z' + w_{j_l}$.
Retornar al Paso 2.

- Paso 4: Hacer $j = j + 1$. Si $j > n$ hacer $t = t + 1$, $\Delta_t = MaxCap_{m(t-1)}$. Si $t > T$ hacer $j = 0, t = 2$ e ir al paso 5, sino si $i_j < m, y_{i_j jt} = 1, Cap_{i_l(t-1)} + r_{i_j jt} \leq MaxCap_{i_j}$ y $r_{i_j jt} \leq \Delta_t$, hacer $W_{i_j(t-1)} = W_{i_j(t-1)} + r_{i_j jt}, MaxCap_{mt} = MaxCap_{mt} + r_{i_j jt}$, $\Delta_t = \Delta_t - r_{i_j jt}$, $Cap_{i_l(t-1)} = Cap_{i_l(t-1)} + r_{i_j jt}$ y repetir el paso.

- Paso 5: Hacer $j = j + 1$. Si $j > n$ hacer $t = t + 1$ Si $t > T$ terminar, sino si $i_j < m$, $y_{i_j jt} = 0, Cap_{i_j t} + r_{i_j jt} \leq MaxCap_{i_j}$ y $r_{i_j jt} \leq MaxCap_{mt}$ hacer $y_{i_j jt} = 1, W_{i_j t} = W_{i_j i_j t} + r_{i_j jt}$, $MaxCap_{mt} = MaxCap_{mt} - r_{i_j jt}$, $Cap_{i_j t} = Cap_{i_j t} + r_{i_j jt}$, $Z' = Z' + w_j$ y repetir el paso.

Ahora, conviene estudiar la complejidad del algoritmo RS.

- ***Teorema 15:*** *El algoritmo RS tiene orden de complejidad máximo* $O(mnTLog(nT))$.

Prueba: En el paso 1, se determina que requerimientos son procesables en función de X_{ij} en un tiempo $O(mnT)$, ordenándolos en un tiempo $O(nTLog(nT))$. Los demás pasos corren en tiempo $O(nT)$. Suponiendo constantes $a, b, c > 0$ y que $m < n$ y $m < T$ (La cantidad de agentes suele ser poca en relación a los períodos de tiempo y a los requerimientos) se tiene:

$$a(mnT) + b\big(nTLog(nT)\big) + c(nT) \leq a(mnT) + b\big(mnTLog(nT)\big) + c(mnT)$$
$$= mnT(a + bLog(nT) + c) \leq mnT\big(aLog(nT) + bLog(nT) + cLog(nT)\big)$$
$$= mnTLog(nT)(a + b + c) = O(mnTLog(nT))$$

3.3.2 CÁLCULO DEL NÚMERO INICIAL DE INSTANCIAS

Ahora, debe calcularse, mediante una fundamentación estadística adecuada, el número de instancias para corroborar γ. La Prueba de Hipótesis a verificar durante el experimento es:

$$H_0: p = 0{,}995$$
$$H_1: p < 0{,}995$$

Es decir, se asume como hipótesis que el algoritmo RS encuentra soluciones factibles (en caso de que existan) para el DGAP el 99,5% de las veces, a partir del intento de minimización de [35], bajo el supuesto de que X_{ij} *es conocido* $\forall\, i, j$. Se trabajará con una confiabilidad del 95% y una potencia deseada del 95% de detectar una probabilidad real $p \leq 0{,}95$, con lo que $\delta = 0{,}045$. Del teorema 14, deben encontrarse K y K' tales que $\sum_{k=0}^{K'-1} \binom{K}{k} (1-\gamma)^k \gamma^{K-k} = \alpha$ y $\sum_{k=K'}^{K} \binom{K}{k} (1-\gamma-\delta)^k (\gamma+\delta)^{K-k} = \beta$. Hacer esto de manera exhaustiva puede traer consigo un gasto computacional muy grande, ya que, K puede llegar a ser demasiado grande. En lugar de ello, se parte del supuesto de aproximación a la distribución normal, bajo la hipótesis que $K(1-\gamma) \geq K\gamma \geq 5$ y $K(1-\gamma-\delta) \geq K(\gamma+\delta) \geq 5$. Así, se tendría lo siguiente:

$$Z_{1-\alpha} = \frac{K' - K(1-\gamma)}{\sqrt{K(1-\gamma)\gamma}} \qquad [36]$$

$$Z_{\beta} = \frac{K' - K(1-\gamma-\delta)}{\sqrt{K(1-\gamma-\delta)(\gamma+\delta)}} \qquad [37]$$

Despejando K' de ambas ecuaciones:

$$K' = Z_{1-\alpha}\sqrt{K(1-\gamma)\gamma} + K(1-\gamma) = Z_{\beta}\sqrt{K(1-\gamma-\delta)(\gamma+\delta)} + K(1-\gamma-\delta)$$

$$\sqrt{K}\left(Z_{1-\alpha}\sqrt{(1-\gamma)\gamma} - Z_{\beta}\sqrt{(1-\gamma-\delta)(\gamma+\delta)}\right) = K(-\delta) \qquad [38]$$

Despejando K se obtiene:

$$K = \frac{\left(Z_{1-\alpha}\sqrt{(1-\gamma)\gamma} - Z_{\beta}\sqrt{(1-\gamma-\delta)(\gamma+\delta)}\right)^2}{\delta^2} \qquad [39]$$

Para el caso de interés, $Z_{0,95} = -1{,}645$ y $Z_{0,05} = 1{,}645$. Sustituyendo en [39]:

$$K = \frac{\left((-1{,}645)\sqrt{(0{,}995)(0{,}005)} - (1{,}645)\sqrt{(0{,}95)(0{,}05)}\right)^2}{0{,}045^2} \approx 111$$

Dado que el anterior resultado es solo una medida de aproximación, se toma como semilla inicial para generar un entorno de búsqueda, que se seleccionó entre 90 y 120. El algoritmo de búsqueda empleado fue el siguiente, implementado en lenguaje Visual Basic para Excel, debido a la facilidad para el cálculo de las probabilidades binomiales:

- Paso 1: Hacer $i = 90, j = 1$.
- Paso 2: Si $\sum_{k=0}^{j-1} \binom{i}{k} (1-\gamma)^k \gamma^{i-k} \geq \alpha$ y $\sum_{k=j}^{i} \binom{i}{k} (1-\gamma-\delta)^k (\gamma+\delta)^{i-k} \leq \beta$ hacer $K = i$ y $K' = j$ y terminar, sino hacer $j = j + 1$. Si $j \leq i$ repetir 2, sino hacer $i = i + 1$. Si $i \leq 120$ repetir 2, sino terminar.

Al momento de ejecutar el código, se obtuvo $K = 120$ y $K' = 118$. Este último resultado indica que, en las 90 instancias que se deben ejecutar, al menos en 118 el algoritmo RS debe encontrar el 99,5% de sus soluciones factibles.

3.3.3 GENERACIÓN DE INSTANCIAS INICIALES

Las instancias se generaron de manera aleatoria, de acuerdo a la siguiente metodología:

- Número de agentes: $m = \mathrm{U}(5{,}20)$.
- Número de tareas: $n = \mathrm{U}(100{,}500)$.
- Períodos de tiempo: $T = \mathrm{U}(20{,}100)$.
- Factores de ponderación: $w_i = \mathrm{U}(1{,}100)$.

- Capacidad agente principal: $MaxCap_m = \mathrm{U}(100000{,}5000000)$.
- Capacidad agentes secundarios: $MaxCap_i = MaxCap_m * \frac{0{,}2mk_i}{\sum k_i}$. Donde $k_i = \mathrm{U}(1{,}5)$.
- Requerimiento neto período t: $\frac{\sum_{i,j} r_{ijt}}{n} = MaxCap_m * \frac{0{,}7Tk_t}{\sum k_t}$. Donde $k_t = \mathrm{U}(1{,}100000)$.
- Requerimiento medio por tarea en t: $r_{jt} = \sum_j r_{ijt} * \frac{k_t}{\sum k_t}$. Donde $k_t = \mathrm{U}(1{,}10000000)$.
- Solución inicial: $: \forall\, j: i = \mathrm{U}(1, m)/X_{ij} = 1$.

Se utiliza la distribución uniforme debido a que en la literatura es la más usada para el desarrollo de instancias en problemas de optimización. De hecho la mayoría de parámetros, exceptuando posiblemente a la demanda de las tareas, tienen valores que son igualmente probables en un entorno generalizado.

3.3.4 RESOLUCIÓN DE INSTANCIAS INICIALES

Las instancias para [35] fueron generadas mediante código Visual Basic de Excel, y fueron resueltas de manera óptima con el software GAMS 23.6, mediante la implementación del solver CPLEX. Por otra parte, el algoritmo RS fue codificado e implementado en el software MATLAB 7.9.0. Todos los softwares empleados fueron ejecutados en computadores DELL PRECISION con procesador Core i5 y 2 GB de memoria Ram. Los resultados se ilustran a continuación, teniendo en cuenta que la función a maximizar es $\sum_{i,j,t/r_{ijt}>0} \omega_j y_{ijt}$:

$Instancia$	Z_{max}	Z_{RS}	Z_{RS}/Z_{max}
R001	985544	983817	0,9982
R002	175919	175909	0,9999
R003	983598	983407	0,9998
R004	1085763	1085289	0,9996

R005	1759701	1757975	0,9990
R006	1598814	1598548	0,9998
R007	425529	425173	0,9992
R008	2011717	2008167	0,9982
R009	920204	919802	0,9996
R010	1792785	1791768	0,9994
R011	826384	825474	0,9989
R012	1275849	1275048	0,9994
R013	1136058	1135313	0,9993
R014	1376654	1375853	0,9994
R015	1037893	1037140	0,9993
R016	520382	519757	0,9988
R017	259211	259154	0,9998
R018	1066788	1065436	0,9987
R019	823426	822326	0,9987
R020	1681011	1680304	0,9996
R021	682208	681954	0,9996
R022	807887	807323	0,9993
R023	398185	397969	0,9995
R024	1333011	1332140	0,9993
R025	381893	381776	0,9997
R026	1836320	1832879	0,9981
R027	2054185	2051090	0,9985
R028	718201	717321	0,9988
R029	962549	962093	0,9995
R030	847308	846790	0,9994
R031	1428657	1427634	0,9993
R032	997596	996369	0,9988
R033	984874	984361	0,9995
R034	1768595	1767631	0,9995
R035	510401	510012	0,9992
R036	446914	446864	0,9999
R037	768997	768902	0,9999
R038	891292	890602	0,9992
R039	1272268	1270090	0,9983
R040	1237688	1237248	0,9996
R041	921368	919804	0,9983
R042	970476	969269	0,9988
R043	331114	330905	0,9994
R044	523266	522600	0,9987

R045	1928647	1928111	0,9997
R046	1515529	1514901	0,9996
R047	2051061	2049514	0,9992
R048	889979	889478	0,9994
R049	1633870	1629846	0,9975
R050	887928	887472	0,9995
R051	1925294	1924498	0,9996
R052	1834274	1833148	0,9994
R053	761986	761309	0,9991
R054	1113040	1112164	0,9992
R055	373633	373468	0,9996
R056	2043837	2042422	0,9993
R057	760358	759721	0,9992
R058	1021028	1020532	0,9995
R059	257173	256832	0,9987
R060	1251433	1250419	0,9992
R061	1194082	1192268	0,9985
R062	350638	349615	0,9971
R063	456646	455535	0,9976
R064	384669	384326	0,9991
R065	594810	594446	0,9994
R066	1115281	1113484	0,9984
R067	1789091	1785689	0,9981
R068	1407153	1404173	0,9979
R069	598580	596564	0,9966
R070	310450	309327	0,9964
R071	952713	951713	0,9990
R072	611193	608892	0,9962
R073	1221970	1220832	0,9991
R074	340105	339860	0,9993
R075	290545	289848	0,9976
R076	520419	519481	0,9982
R077	1419440	1413006	0,9955
R078	714495	712244	0,9968
R079	1193359	1192532	0,9993
R080	1870453	1867127	0,9982
R081	548296	547480	0,9985
R082	840854	839712	0,9986
R083	1303720	1302881	0,9994
R084	731281	728207	0,9958

R085	874781	873876	0,9990
R086	2071966	2070380	0,9992
R087	906312	904240	0,9977
R088	641033	640364	0,9990
R089	788770	786968	0,9977
R090	695245	694576	0,9990
R091	419407	419305	0,9998
R092	1323459	1320392	0,9977
R093	680348	679850	0,9993
R094	1887762	1886239	0,9992
R095	1889968	1885222	0,9975
R096	500105	499448	0,9987
R097	1002151	1001333	0,9992
R098	441204	440813	0,9991
R099	602007	600255	0,9971
R100	878669	871373	0,9917
R101	268879	268570	0,9989
R102	694195	693644	0,9992
R103	759710	757508	0,9971
R104	926692	925184	0,9984
R105	701876	699883	0,9972
R106	1963477	1962231	0,9994
R107	1065189	1064281	0,9991
R108	414531	414264	0,9994
R109	2068882	2067161	0,9992
R110	415346	414852	0,9988
R111	828885	824130	0,9943
R112	773969	771291	0,9965
R113	1018021	1016080	0,9981
R114	206545	206384	0,9992
R115	1133064	1132354	0,9994
R116	305385	305085	0,9990
R117	547347	546816	0,9990
R118	250287	249320	0,9961
R119	630529	629983	0,9991
R120	1229973	1228687	0,9990

De los resultados se tiene que el Algoritmo RS encontró el 99,5% o más del total de soluciones factibles para 118 de las 120 instancias, o lo que es lo mismo, en 118 de las 120 instancias se tuvo un desempeño del 99,5% o más respecto a la solución óptima. Como $K' = 118$, no se puede rechazar la hipótesis nula, por lo que, con una confiabilidad del 95%, se concluye que el DGAP es de orden parcialmente polinomial al nivel 0,995, o lo que es equivalente, existe una probabilidad del 95% de que el Algoritmo RS encuentre soluciones factibles para el DGAP el 99,5% de las veces, mediante la maximización de la restricción de máximo factible expresada en [33] y [35]. Además de lo anterior, existe una probabilidad del 5% (error tipo II) de haber tomado una mala decisión si el verdadero valor de $1-\gamma$ fuese 0,95 o menor.

3.4 FASE DE EXPERIMENTACIÓN 2

Hasta ahora todo este capítulo ha sido dedicado a probar que el DGAP es de orden parcialmente polinomial al nivel 0,995, pero ¿Qué repercusión tiene esto en la presente investigación?. En realidad, tiene una repercusión enorme.

El objetivo principal de esta investigación es proponer un algoritmo genético "mejorado" para resolver el DGAP. Sin embargo, está probado en la literatura que los algoritmos genéticos experimentan problemas para resolver modelos matemáticos, debido a la necesidad de cumplir restricciones, lo que daría al traste con los deseos de este trabajo. Las variables a codificar serían:

- Variables X_{ij}: mn.
- Variables y_{ijt}: mnT.
- Variables Cap_{it}: mT.
- Variables W_{it}: mT.

El total de variables a codificar sería entonces $mn + mnT + 2mT$. Además, se deben satisfacer todas las restricciones, que son en total $n + mnT + 3nT + T + 1$.

Lo anterior claramente representa un desafío para cualquier algoritmo genético que se desee emplear. Sin embargo, gracias a que el DGAP posee un orden parcialmente polinomial alto, la cantidad de variables en la codificación del algoritmo genético puede reducirse a mn, considerando solo una restricción.

3.4.1 ALGORITMO GENÉTICO POTENCIADO (EGA)

El Algoritmo Genético propuesto en este trabajo para resolver el DGAP se denomina Algoritmo Genético Potenciado (EGA). La frase "Potenciado" hace alusión a un mejoramiento especial hecho sobre el algoritmo para resolver el DGAP. Dicho mejoramiento está compuesto por dos fases: Reducción de la cantidad de variables y restricciones y el establecimiento de modificaciones sobre los operadores de selección, cruzamiento y mutación. El EGA es elitista, lo que implica que en cada generación siempre se guarda el mejor de los individuos.

3.4.1.1 REDUCCIÓN DEL NÚMERO DE VARIABLES Y RESTRICCIONES

Gracias al hecho de que el DGAP es de orden parcialmente polinomial al nivel 99,5%, se puede modificar la codificación del algoritmo genético de modo que se reduzca drásticamente la cantidad de variables. Suponga que durante el algoritmo solo se codificaran las variables X_{ij}. La pregunta obvia que resulta es: ¿y el resto de las variables?. Codificar solo X_{ij} implica que todas estas variables serán conocidas, y dado que el DGAP es de orden parcialmente polinomial al nivel 99,5%, puede usarse el algoritmo RS para resolver [35], encontrando en tiempo fuertemente polinomial (baja complejidad del algoritmo) el resto de variables, garantizando el cumplimiento de casi todas las restricciones, exceptuando, posiblemente, la restricción del nivel de servicio. Sin embargo, ya se demostró que existe una probabilidad de al menos 0,995 de que la restricción de máximo factible, sea como

tal, factible. Por tanto, solo es necesario que cada individuo tenga mn genes, con lo que la cantidad de variables a codificar se reduce en:

$$\frac{mn + mnT + 2mT - mn}{mn + mnT + 2mT} = \frac{mnT + 2mT}{mn + mnT + 2mT} = \frac{(n+2)T}{n + (n+2)T} \qquad [40]$$

Para n lo suficientemente grande, se tendrá que $n \approx n + 2$ por lo que:

$$\frac{(n+2)T}{n + (n+2)T} \approx \frac{nT}{n + nT} = \frac{T}{1 + T} \qquad [41]$$

Así, si se tuviesen 4 instantes de tiempo, la reducción en la cantidad de variables a codificar sería del 100%, con 10 instantes 90,9%, y con 20 95,23%.
En lo que concierne a la cantidad de restricciones, solo existe una probabilidad de 0,005 de que no se satisfaga la restricción de máximo factible. La reducción en la cantidad de restricciones a considerar es:

$$\frac{n + mnT + 3nT + T + 1 - 1}{n + mnT + 3nT + T + 1} = \frac{n + mnT + 3nT + T}{n + mnT + 3nT + T + 1} \qquad [42]$$

Dado que en [42] el numerador es aproximadamente igual al denominador en la mayoría de los casos, se tendría una reducción en el número de restricciones superior al 90%. Por tanto, un Algoritmo Genético basado en este nuevo esquema de codificación resultaría ser extremadamente eficiente, convergiendo hacia el óptimo a una tasa superior a la normal (siempre y cuando los operadores sean adecuadamente definidos).
La codificación para cada cromosoma consistirá de n genes (uno para cada tarea), donde cada gen podrá tomar valores entre 1 y m (número de agentes).

3.4.1.2 MODIFICACIÓN DE OPERADORES GENÉTICOS

Hasta el momento se ha creado un compendio teórico que permitirá mejorar la eficiencia del Algoritmo Genético a usar para tratar el DGAP. Sin embargo, en lo que concierne a la calidad de las respuestas son los operadores los que juegan el rol más importante. A continuación se definen los operadores clave para el EGA.

3.4.1.2.1 SELECCIÓN

Se considerará una selección por ruleta del resto con reemplazo. La razón por la que se escoge esta forma para el operador de selección es debido a la necesidad de introducir variedad en las soluciones, minimizando la probabilidad de que se converja prematuramente a óptimos locales. Sin embargo, se introduce una modificación especial respecto al método tradicional.

En caso de que existan muchos individuos en la población, la porción de ruleta para cada cromosoma será relativamente pequeña, con lo que indistintamente todos los individuos tendrían probabilidades no muy distantes de ser seleccionados. Sean $\bar{A}$ y σ_A la aptitud promedio y desviación de la aptitud en la población, respectivamente. Para cada individuo s, se calcula el número de desviaciones estándar como:

$$k_s = \frac{a_s - \bar{A}}{\sigma_A} \qquad [43]$$

Luego, se calcula la aptitud corregida para cada individuo como:

$$a'_s = a_s + \lambda\sigma_A k_s = a_s + \lambda\sigma_A \frac{a_s - \bar{A}}{\sigma_A} = (1+\lambda)a_s - \lambda\bar{A} \quad [44]$$

Donde:

$$\lambda \leq \frac{a_{s'}}{\bar{A} - a_{s'}};\ \ s' : \frac{a_{s'}}{\bar{A} - a_{s'}} \leq \frac{a_s}{\bar{A} - a_s};\ \bar{A} - a_{s'} > 0 \ \wedge\ \bar{A} - a_s > 0 \quad [45]$$

Lo anterior permite generar mayor dispersión entre las aptitudes de los individuos, incrementando la probabilidad de que los individuos más prometedores sean seleccionados.

3.4.1.2.2 CRUZAMIENTO

El cruzamiento elegido para el EGA es de un único punto. Sin embargo, tiene una modificación importante. Sea $l < n$ un tramo del cromosoma, delimitado por los genes j' y j'', tales $j' < j''$ y $l = j'' - j' + 1$. De entre los dos individuos candidatos para el cruzamiento, se selecciona el tramo l que resulte más costoso. Sean p_{c1} y p_{c1} , $p_{c1} + p_{c2} < 1$, dos probabilidades distintas de cruzamiento, las cuales se sortean por ruleta. Si resulta elegido el cruzamiento 1, se genera un aleatorio entre 1 y $n - 1$ para determinar después de que gen efectuar el cruzamiento. Si resulta elegido el cruzamiento 2, se genera un aleatorio entre j' y $j'' - 1$ para determinar el punto de cruzamiento, donde dichos puntos corresponden al tramo más costoso de longitud l entre los dos individuos.

3.4.1.2.3 MUTACIÓN

El papel principal de la mutación es evitar caer en óptimos locales, por lo que aquí se introduce una modificación importante en este operador. Se introduce el término de persistencia, el cual consiste en determinar una función que mida el grado de aparición de un agente i en un gen j. Dicha función se define como:

$$Persistencia(i,j) = \frac{\tau_{ij}}{g} \qquad [46]$$

Siendo τ_{ij} el número de veces que el agente i apareció en el gen j del mejor cromosoma, y g el total de generaciones corridas, con $G \geq g$ el total de

generaciones del algoritmo. Luego, al ser p_m la probabilidad a priori de mutación, la probabilidad a posteriori de que el agente i presente en el gen j mute es:

$$p'_m = p_m * Persistencia(i,j) \quad [47]$$

Así, [47] permite dar mayor probabilidad de mutación a los individuos más persistentes, evitando caer así en óptimos locales.

3.4.1.3 TRATAMIENTO DE SOLUCIONES NO FACTIBLES

Dado que existe una probabilidad del 0,5% de que el algoritmo RS no encuentre una solución factible para [35], debe amoldarse cualquier aptitud (valor recíproco de la función objetivo) a la no factibilidad, o dicho en otras palabras, debe penalizarse.
La gran mayoría de autores hacen penalizaciones lineales en función del desbordamiento de las restricciones. Esto puede ser inadecuado debido a que el desbordamiento puede ser pequeño en relación a la aptitud. Así, en lugar de ellos, en este trabajo se propone una penalización multiplicativa, la cual no dependerá de la diferencia entre el desbordamiento y la aptitud:

$$a_{s(corregido)} = \begin{cases} a_s \text{ , si } \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - y_{ijt}) \leq U_\omega \\ \dfrac{a_s \sum_{(i,j,t)/r_{ijt}>0;}\ \omega_j(X_{ij} - y_{ijt})}{U_\omega}, si \sum_{(i,j,t)/r_{ijt}>0;} \omega_j(X_{ij} - y_{ijt}) > U_\omega \end{cases}$$

[48]

3.4.1.4 PSEUDOCÓDIGO DEL EGA

Para la ejecución del EGA, se asumen conocidos los parámetros de entrada $l, p_{c1}, p_{c2}, p_m, P$ y G. El parámetro λ es ajustable dependiendo de la generación que

se esté analizando, mientras que las P soluciones iniciales se generan de manera aleatoria. El esquema del algoritmo es el siguiente:

- Paso 1: Generar la población inicial. A cada solución aplicarle el algoritmo RS y determinar los $a_{s(corregido)}$. Hacer $g = 1$, guardar la mejor solución e ir al paso 2.

- Paso 2: Hacer $g = g + 1$. Si $g > G$ entonces terminar, sino aplicar los operadores de selección, cruzamiento y mutación. A cada individuo aplicarle el algoritmo RS y determinar los $a_{s(corregido)}$. Conservar el mejor individuo entre la generación actual y la anterior, y luego repetir el paso.

Ahora, resulta conveniente analizar la complejidad del EGA:

- ***Teorema 16:*** *El algoritmo EGA tiene orden de complejidad máximo* $O((mnTGP)Log(mnT))$.

Prueba: En primer lugar, todos los mnT pueden ordenarse según la regla voraz en un número de pasos $O(mnTLog(mnT))$. Luego, para cada individuo de la población en una generación determinada, se determina la lista de requerimientos a atender en función de los valores X_{ij} conocidos. Esto se hace en una cantidad de pasos $O(mnTP)$. La selección se hace en un orden máximo $O(P^2)$, el cruzamiento , la mutación y el cálculo de la aptitud corregida requieren de un número de pasos de orden máximo $O(nP)$. Lo anterior se hace para todas las G generaciones.
Asumiendo que $P^2 < mnTP$ para problemas moderadamente grandes, se tiene:

$$a_1 mnTLog(mnT) + a_2 mnTGP + a_3 GP^2 + a_4 nGP$$
$$\leq a_1 mnTGLog(mnT) + a_2 mnTGPLog(mnT) + a_3 mnTGPLog(mnT)$$
$$+ a_4 mnTGPLog(mnT)$$
$$\leq (a_1 + a_2 + a_3 + a_4) mnTGPLog(mnT) = O(mnTGPLog(mnT))$$

3.4.2 CÁLCULO DEL NÚMERO DE INSTANCIAS PARA EL DGAP

El procedimiento es similar al planteado en la sección 3.3.2. Se considera que no es adecuado utilizar el promedio muestral como variable aleatoria, ya que no se establece una garantía mínima de desempeño. En lugar de ello, al igual que en el caso del algoritmo RS, se empleará una prueba no paramétrica basada en un experimento Bernoulli. Como hipótesis, se establece que la probabilidad de que la máxima desviación respecto al óptimo sea 0,01 es del 99%:

$$H_0: p = 0{,}99$$
$$H_1: p < 0.99$$

Se asume como peligroso el hecho de que se concluya que la probabilidad es del 99% cuando en realidad es del 94% o menor. Utilizando [39], y asumiendo errores tipo I y tipo II iguales al 5%, se necesita un tamaño mínimo de muestra igual a 122. Tomando como semilla el valor anterior, se llega a que el tamaño mínimo real de instancias debe ser de 125.

3.4.3 GENERACIÓN DE INSTANCIAS PARA EL DGAP

La generación de las instancias para el DGAP se basa en el mismo procedimiento usado con [35], solo que en este caso las X_{ij} no son parámetros. Además, los costos se generaron de la siguiente manera:

- Costos de asignación: $C_{ij} = U(10000{,}100000)$.

3.4.4 RESOLUCIÓN DE INSTANCIAS PARA EL DGAP

El procedimiento para la resolución de instancias del DGAP fue el mismo planteado en la sección 3.3.4. Para el EGA, se utilizó la siguiente parametrización:

- $\lambda^* = 0{,}5\lambda$
- $p_{c1} = 0{,}3$
- $p_{c1} = 0{,}65$
- $p_m = 0{,}025$
- $P = 20$
- $G = 100$

Los resultados obtenidos fueron los siguientes:

$Instancia$	Z_{OPT}	$t - Z_{OPT}\ (seg)$	Z_{EGA}	$t - Z_{EGA}\ (seg)$	$\frac{Z_{EGA} - Z_{OPT}}{Z_{OPT}}$
EGA001	2538737	29047,21	2552474	844,39	0,0054
EGA002	3496484	18159,87	3519668	841,94	0,0066
EGA003	6200007	25457,15	6236960	467,76	0,0060
EGA004	3512884	24458,62	3535750	862,54	0,0065
EGA005	3363920	16898,84	3386942	693,07	0,0068
EGA006	4104119	13851,67	4136585	761,42	0,0079
EGA007	3969427	12246,57	3980434	601,81	0,0028
EGA008	9434428	12802,74	9536628	198,17	0,0108
EGA009	4852973	19098,33	4866065	800,25	0,0027
EGA010	4308071	19029,2	4321116	174,74	0,0030
EGA011	10872580	12736,9	10927831	885,54	0,0051
EGA012	3507847	20205,41	3524358	291,53	0,0047
EGA013	8209249	6998,76	8267339	228,48	0,0071
EGA014	6761114	24343,37	6801035	128,11	0,0059
EGA015	7079638	29131,26	7087674	196,3	0,0011
EGA016	3120255	11076,81	3143187	508,33	0,0073
EGA017	6589346	9305,07	6641636	753,62	0,0079
EGA018	3972987	26058,38	3988152	455,31	0,0038
EGA019	3414460	27517,35	3436888	882,53	0,0066
EGA020	3690914	7548,06	3720520	428,56	0,0080
EGA021	6934718	28509,72	6974364	523,73	0,0057
EGA022	10834907	24427,02	10889600	201,61	0,0050

EGA023	7026675	24581,01	7031429	972,78	0,0007
EGA024	5615369	7196,73	5654208	917,71	0,0069
EGA025	3086066	28656,17	3094274	749,72	0,0027
EGA026	6706410	9181,07	6745454	624,23	0,0058
EGA027	5075335	24993,75	5083820	798,41	0,0017
EGA028	3975626	26019,5	3987416	725,45	0,0030
EGA029	1912076	10544,85	1921213	603,15	0,0048
EGA030	4177764	16768,39	4184137	204,14	0,0015
EGA031	2249914	29638,98	2260266	120,51	0,0046
EGA032	3276977	7979,77	3296502	465,6	0,0060
EGA033	8362944	7420,42	8434493	183,31	0,0086
EGA034	5863002	27977,22	5866404	488,12	0,0006
EGA035	5342163	10429,36	5351924	818,09	0,0018
EGA036	7340960	17236,88	7384053	450,15	0,0059
EGA037	2432495	29678,77	2433635	261,27	0,0005
EGA038	4278916	28340,37	4297172	679,79	0,0043
EGA039	4973855	18875,63	5015969	633,09	0,0085
EGA040	7664970	7542,86	7707470	688,46	0,0055
EGA041	4830691	14751,32	4840589	533,42	0,0020
EGA042	7395410	20563,18	7454137	955,44	0,0079
EGA043	6272542	23248,45	6313692	859,09	0,0066
EGA044	4636638	21523,59	4642354	789,18	0,0012
EGA045	1983775	29589,88	1990679	962,39	0,0035
EGA046	3260749	18443,92	3276162	278,53	0,0047
EGA047	6061781	25792,41	6085089	481,93	0,0038
EGA048	8332438	28520,26	8348077	299,46	0,0019
EGA049	8391345	12300,87	8459774	824,6	0,0082
EGA050	4331161	7732,18	4334054	178,32	0,0007
EGA051	2683035	27898,28	2688525	926,56	0,0020
EGA052	7754607	17729,45	7789172	906,1	0,0045
EGA053	3917428	8990,99	3929673	610,91	0,0031
EGA054	6167121	23840,12	6189830	742,63	0,0037
EGA055	7858762	13772,98	7869813	274,49	0,0014
EGA056	4846334	27470,01	4858913	714,5	0,0026
EGA057	6660413	22784,17	6680202	566,04	0,0030
EGA058	6232150	12494,46	6253590	503,64	0,0034
EGA059	3888839	7275,66	3906536	795,48	0,0046
EGA060	6782187	26347,92	6815535	194,68	0,0049
EGA061	4307236	14381,99	4342907	889,98	0,0083
EGA062	6039026	25368,88	6074339	712,64	0,0058

EGA063	6292218	8570,44	6324343	378,46	0,0051
EGA064	4472103	14321,3	4477535	641,67	0,0012
EGA065	5179075	9183,75	5212480	157,73	0,0064
EGA066	4939074	16287,66	4948728	222,62	0,0020
EGA067	2872836	15474,51	2886445	540,66	0,0047
EGA068	4835531	25811,75	4841061	925,38	0,0011
EGA069	3135127	29748,79	3152206	984,14	0,0054
EGA070	8556088	28712,22	8578899	412,78	0,0027
EGA071	7879472	20621,35	7930122	590,74	0,0064
EGA072	6131241	12742,76	6154148	471,43	0,0037
EGA073	4113405	26078,53	4143537	799,9	0,0073
EGA074	4638390	24005,26	4676347	797,23	0,0082
EGA075	7953498	26821,58	7973008	308,33	0,0025
EGA076	3346562	9435,45	3351668	226,88	0,0015
EGA077	3127666	25010,27	3141602	165,88	0,0045
EGA078	6865832	8155,89	6924242	489,36	0,0085
EGA079	6268098	28127,87	6289724	359,66	0,0035
EGA080	5731775	14908,1	5777371	162,31	0,0080
EGA081	7130520	15091,5	7136165	231,43	0,0008
EGA082	3097984	15218,6	3102138	464,03	0,0013
EGA083	3090457	5095,15	3102138	889,1	0,0038
EGA084	3213196	24249	3220502	973,24	0,0023
EGA085	4174598	8716,08	4194881	566,8	0,0049
EGA086	10012754	20670,17	10050604	751,47	0,0038
EGA087	10102355	5299,55	10150294	544,75	0,0047
EGA088	7531223	12311,4	7558879	917,77	0,0037
EGA089	7971681	20816,19	7975474	726,82	0,0005
EGA090	4378463	9431,26	4404678	268,55	0,0060
EGA091	5759009	18101,11	5921550	446,63	0,0282
EGA092	10146378	6935,69	10190820	506,27	0,0044
EGA093	7107416	21171,85	7121022	376,52	0,0019
EGA094	3722631	17543,22	3729891	884,25	0,0020
EGA095	5931219	11792,34	5945001	400,65	0,0023
EGA096	2162095	6289,9	2171031	953,91	0,0041
EGA097	6855804	19756,78	6903085	511,41	0,0069
EGA098	7538478	9843,28	7557658	942,43	0,0025
EGA099	3594144	10907,62	3608477	267,71	0,0040
EGA100	9570336	16616,7	9602992	457,97	0,0034
EGA101	6421413	16523,72	6465102	950,91	0,0068
EGA102	3883692	24192,49	3893509	993,61	0,0025

EGA103	6514374	5775,75	6540390	755,98	0,0040
EGA104	4119268	18347,4	4145789	771,84	0,0064
EGA105	4383362	26706,79	4386094	781,19	0,0006
EGA106	4697159	29774	4712239	317,68	0,0032
EGA107	5774380	6394,68	5779871	969,11	0,0010
EGA108	6812957	13927,82	6845192	331,9	0,0047
EGA109	6731295	29812,41	6761010	456,56	0,0044
EGA110	10419751	5731,06	10467663	361,93	0,0046
EGA111	6310915	14122,5	6351620	374,46	0,0064
EGA112	6290597	17602,55	6343771	481,02	0,0085
EGA113	10052370	16247,98	10131459	485,36	0,0079
EGA114	5979195	6346,87	6030880	862,01	0,0086
EGA115	6680146	28996,96	6721485	891,64	0,0062
EGA116	3776212	6531,75	3783977	236	0,0021
EGA117	7405144	24823,49	7435729	619,51	0,0041
EGA118	8655976	17898,44	8671025	244,23	0,0017
EGA119	7011463	5817,97	7029388	763,04	0,0026
EGA120	6106872	22327,11	6138594	489,34	0,0052
EGA121	2350770	25405,51	2371243	311,49	0,0087
EGA122	8313355	20218,11	8353084	849,76	0,0048
EGA123	7581776	20289,31	7615487	560,4	0,0044
EGA124	7487824	18942,77	7514645	510,42	0,0036
EGA125	10440987	28899,81	10490628	214,28	0,0048

De los resultados, solo dos instancias tienen una desviación mayor al 1%. El p-valor de la prueba es entonces:

$$p - valor = \sum_{i=0}^{123} \binom{125}{i} (0{,}99^{i})(0{,}01)^{125-i} = 0{,}3558 > 0{,}05$$

Así, no se puede rechazar la hipótesis nula, con lo que se concluye con una confiabilidad del 95%, que el 99% de las veces EGA tiene una desviación máxima del 1% respecto al óptimo.

Además del excelente desempeño que presenta el EGA en cuanto a calidad de respuesta, resulta ser también muy eficiente, ya que, en cuestión de pocos minutos,

resuelve instancias para las que encontrar la solución óptima se tardaría horas o días de cómputo.

4 CASO DE ESTUDIO: COMPAÑÍA GLOBAL DE PINTURAS

El DGAP nació de un problema de la vida real. Durante la ejecución de un proyecto de investigación en la red Mundial Logistic Service (MLS), se necesitaba rediseñar la red de distribución de la empresa Compañía Global de Pinturas (CGP), implicando esto determinar que clientes en el país debían ser atendidos por cada uno de los centros de distribución, de modo que el costo total de atención fuera mínimo y el nivel de servicio fuese de al menos 95%. Claramente, se trataba de un modelo de asignación.

Se modeló el sistema de manera estratégica, considerando solo las 30 ciudades más importantes, representando estas el 95% de toda la demanda. Se estudió la demanda en galones de pintura para los años 2009 y 2010. A saber, los 6 centros de distribución de CGP, así como su capacidad diaria de almacenamiento en galones de pintura son:

Cedi	Capacidad diaria en galones
G21 (Barranquilla)	6000
G22 (Funza)	12000

G23 (Bucaramanga)	7000
G24 (Cali)	8000
G26 (Pasto)	1000
G28 (Rionegro)	41000

Rionegro es el Cedi principal, encargado de distribuir a los 5 cedis restantes. Para cualquier caso, resultaba más económico atender cualquier ciudad directamente desde Rionegro, ya que no se incurría en costo de cargue, descargue y almacenamiento respecto al cedi intermedio o secundario. Como el modelo inicial era estratégico (Demanda agregada), el modelo de transporte sugerido al inicio del proyecto arrojaba como solución que todos los clientes debían ser atendidos desde Rionegro, ya que, la capacidad agregada así lo permitía. Dichos resultados no convencieron a los directivos de CGP ya que, para ellos, se trataba de algo ilógico e irreal. Después de un arduo análisis, se detectó algo que no estaba siendo tenido en cuenta, y era el hecho de que G28 era el cuello de botella del sistema, o dicho en otras palabras, existían períodos de tiempo en los que G28 no era capaz de atender toda la demanda. Así, no resultaba conveniente un modelo estratégico, era necesario diseñar un modelo táctico-operativo, que tuviera en cuenta la demanda período a período y el nivel de servicio en la atención a los clientes; fue así como surgió el DGAP.

A continuación se describen las dos fases del caso, la fase de levantamiento de información y la fase de ejecución.

4.1 LEVANTAMIENTO DE INFORMACIÓN

Los costos, capacidades y demandas fueron suministrados por CGP. El costo por galón desde cada Cedi a cada ciudad se ilustra a continuación:

Ciudad\Cedi	G21	G22	G23	G24	G26	G28
BARRANCABERMEJA_SAN	3572.7512	1679.69003	1612.26204	3554.12532	5841.60644	1076.1213

BARRANQUILLA_ATL	1744.42808	2152.7916	3206.61621	4013.18782	5769.037	954.159499
BELLO_ANT	3002.66092	1650.70566	2673.95648	2367.63226	4951.06477	331.01282
BOGOTA_BOG	3574.88314	1472.69543	2673.95648	2525.41004	5002.70366	905.512886
BUCARAMANGA_SAN	3572.7512	1679.69003	1199.1744	3554.12532	5841.60644	796.59608
CALI_VAC	4237.48731	1679.69003	3472.22037	1365.07123	4023.01616	895.793735
CALOTO_CAU	4237.48731	1679.28378	3472.22037	1652.07671	4023.01616	1171.73769
CARTAGENA_BOL	2575.64703	2166.6705	3331.43565	3954.70865	5769.037	971.06033
CUCUTA_NSA	3971.88314	2175.65097	2293.69259	4262.4031	6099.95366	1037.08465
DOSQUEBRADAS_RIS	4121.3762	1643.3541	3131.14398	2411.88226	4279.912	1437.91521
ENVIGADO_ANT	3002.66092	1650.70566	2673.95648	2367.63226	4951.06477	381.896083
FUNZA_CUN	3574.88314	1212.97128	2673.95648	2525.41004	5002.70366	872.16689
GIRON_SAN	3572.7512	1679.69003	1612.26204	3554.12532	5841.60644	824.06219
GUARNE_ANT	3217.24426	1650.70566	2673.95648	2510.9656	4951.06477	142.023063
ITAGUI_ANT	3002.66092	1650.70566	2673.95648	2367.63226	4951.06477	446.766063
JAMUNDI_VAC	4237.48731	1679.69003	3472.22037	1632.77115	4023.01616	874.533516
MANIZALES_CAL	4121.3762	1713.56503	3131.14398	2411.88226	4279.912	833.394231
MEDELLIN_ANT	3002.66092	1650.70566	2673.95648	2367.63226	4951.06477	301.691704
MONTERIA_COR	2629.04981	2023.28378	3112.26204	3752.07671	5703.84255	337.552255
PALMIRA_VAC	4237.48731	1679.69003	3472.22037	1632.77115	4023.01616	1059.66557
PASTO_NAR	6447.95259	2503.02988	4203.72037	2896.64615	1901.92471	1301.76077
PEREIRA_RIS	4121.3762	1643.3541	3131.14398	2411.88226	4279.912	746.430245
POPAYAN_CAU	4237.48731	1679.28378	3472.22037	1652.07671	4023.01616	1562.197
RIONEGRO_ANT	3217.24426	1650.70566	2673.95648	2510.9656	4951.06477	208.589351
SANTA MARTA_MAG	2508.88314	2152.7916	3206.61621	4110.00726	5892.40505	1035.86272
SINCELEJO_SUC	2629.04981	2023.28378	3112.26204	3752.07671	5703.84255	997.377929
VALLEDUPAR_CES	2629.04981	2123.28378	3112.26204	3852.07671	5703.84255	1636.33416
VILLAVICENCIO_MET	4304.2512	1600.90097	3554.94954	3578.79893	5854.66894	1605.2757

No se muestra la tabla de demandas ya que son 700 días de demanda considerados. El costo de asignación se calculó multiplicando la demanda de cada

cliente por el costo por galón transportado. Para las ponderaciones w_j se procedió del siguiente modo: La demanda total de cada ciudad en el horizonte de tiempo estudiado se multiplicó por el precio de venta promedio de un galón de pintura. Luego cada total se dividió entre el total más pequeño, calculando así los factores w_j. La suma de dichos factores multiplicada por 0,05 origina el parámetro U_w.

4.2 EJECUCIÓN

Una vez se completó el levantamiento y validación de información, se procedió a implementar el EGA a la instancia resultante. La parametrización usada fue la misma que se describió en la sección 3.4.4. El costo de asignación obtenido fue de $5.784'643.324, con un nivel de servicio del 95,62%. El costo inicial (sin mejora) era de $5.900'000.000 aproximadamente, con un nivel de servicio del 93%.

4.3 CONCLUSIONES CASO DE ESTUDIO

Los resultados obtenidos durante la investigación fueron satisfactorios, ya que el costo total de atención se redujo, incrementando el nivel de servicio. Dichos resultados fueron validados por la empresa, quien espera poder implementar las mejoras propuestas a inicios del año 2013.

Este caso de estudio motivó a la creación de un nuevo modelo matemático de asignación, el cual, tiene muchas otras aplicaciones de nivel práctico.

5 CONCLUSIONES

Esta investigación permitió proponer un nuevo modelo de asignación, denominado Problema de Asignación Generalizado Dinámico (DGAP), el cual tiene en cuenta aspectos tales como nivel de servicio y variación de requerimientos a lo largo del tiempo. Dicho modelo tiene enormes aplicaciones en diversas situaciones de la vida real. Se mostró que el problema aquí tratado pertenece a la familia NP-Duro, por lo que resultaba conveniente proponer una metaheurística que obtuviera soluciones de buena calidad en tiempo razonable. Fue así como se propuso un algoritmo genético modificado, que se aprovecha de varias características del problema para incrementar la eficiencia y la calidad de las respuestas obtenidas.

A diferencia de la gran mayoría de trabajos encontrados en la literatura, donde no se justifica el número de instancias probadas, aquí se determinó de manera estadística la cantidad mínima de instancias a probar, de manera que se garantizaran la confiabilidad y la potencia del test del test de hipótesis propuesto inicialmente. En lugar de usar la usual prueba de medias, se prefirió trabajar con un experimento Bernoulli, ya que la prueba de medias no tiene como resultado un valor numérico que permita medir numéricamente el desempeño del algoritmo.

Los resultados obtenidos muestran que, el DGAP es de Orden Parcialmente Polinomial al nivel 0,995, con una confiabilidad del 95%, pues, existe un algoritmo (RS) que encuentra soluciones factibles, en caso de que existan, para al menos el 99,5% de las instancias. Esta propiedad matemática tan importante, permitió reducir enormemente (en más de un 99%) el total de variables a codificar en el algoritmo genético, lo que reduce drásticamente la probabilidad de aparición de soluciones no factibles, incrementando la eficiencia y la calidad en las respuestas obtenidas.

Las pruebas también mostraron que, con una confiabilidad del 95%, el EGA tiene una desviación máxima del 1% respecto al óptimo el 99% de las veces, en las instancias probadas, por lo que se puede aspirar a obtener soluciones muy cercanas a la óptima en tiempos razonables, como lo ilustran los resultados experimentales.

El modelo fue propuesto con éxito en un entorno de la vida real con 6 agentes, 30 tareas y cerca de 700 instantes de tiempo. Se resolvió dicha instancia mediante EGA. Los resultados obtenidos fueron satisfactorios, y tienen una gran probabilidad de ser aplicados en el sistema estudiado. Gracias a este caso de estudio surgió el nuevo modelo matemático de asignación, que tiene un amplio margen de aplicaciones en entornos de la vida real, donde la metodología aquí propuesta (EGA) puede resultar importante a la hora de obtener soluciones de gran factura en tiempos de cómputo muy pequeños.

Todos los objetivos propuestos en un inicio fueron cumplidos, por lo que la investigación fue completamente exitosa.

6 CONTRIBUCIONES

Las contribuciones más significativas de esta investigación fueron:

- Propuesta de un nuevo modelo de asignación, el cual tiene en cuenta requerimientos cambiantes en el tiempo y nivel de servicio, factores que no son tenidos en cuenta por otros modelos de asignación existentes.

- Demostración matemática que indica que el DGAP es NP-Duro, lo que evita tratar de resolverlo de manera exacta para instancias de gran tamaño, pues para este tipo de problemas no se han podido encontrar algoritmos que encuentren la solución óptima en tiempo polinomial.

- Prueba estadística que muestra que el DGAP es de Orden Parcialmente Polinomial al nivel 0,995 con una confiabilidad del 95%, lo que permite reducir enormemente la cantidad de variables a codificar en un algoritmo genético, disminuyendo a la vez la probabilidad de aparición de soluciones infactibles.

- Prueba estadística que indica que el EGA encuentra soluciones con desviación máxima del 1% respecto al óptimo el 99% de las veces, con una confiabilidad del 95%, a la hora de resolver instancias del DGAP.

- Aplicación de una metodología experimental más conveniente para problemas de prueba de convergencia de algoritmos.

- Aplicación del modelo y algoritmos propuestos en un caso de la vida real, con soluciones validadas y altamente eficientes.

7 FUTURAS INVESTIGACIONES

Dentro de las investigaciones que pueden desprenderse a partir de este trabajo se tienen:

- Propuesta de nuevos algoritmos de solución, ya sean exactos o aproximados.

- Incorporación de nuevos supuestos y restricciones en el modelo.

- Estudio del caso multi-objetivo.

8 REFERENCIAS BIBLIOGRÁFICAS

Amini, M. M., & Racer, M. (1995). A hybrid heuristic for the generalized assignment problem. *European Journal of Operational Research*, *87*(2), 343-348. doi:10.1016/0377-2217(94)00154-5.

A.P. Punnen, Y.P. Aneja, Categorized assignment scheduling: A tabu search approach, Journal of the OperationalResearch Society 44 (7) (1993) 673–679.

A. Schrijver. On the history of combinatorial optimization(till 1960).

A. Shtub, K. Kogan, Capacityplanning bythe dynamic multi-resource generalized assignment problem (DMRGAP), European J. Oper. Res. 105 (1998) 91–99.

A. Volgenant, Linear and semi-assignment problems: A core oriented approach, Computers & Operations Research 23 (10) (1996) 917–932.

Cattrysse DG, Wassenhove LNV. A survey of algorithms for the generalized assignment problem. Eur J Oper Res 1992;60:260–72.

Choy, W. Y., & Sanctuary, B. C. (1998). Using genetic algorithm with a priory knowledge for quantitative NMR signal analysis. *Journal of Chemical Information and Computer Sciences*, *38*, 685–690.

Cohen, R., Katzir, L., & Raz, D. (2006). An efficient approximation for the Generalized Assignment Problem. *Information Processing Letters*, *100*(4), 162-166. doi:10.1016/j.ipl.2006.06.003

C.W. Duin, A. Volgenant, Minimum deviation and balanced optimization: A unified approach, Operations Research Letters 10 (1) (1991) 43–48.

C.W. Zhang, H.L. Ong, An efficient solution to biobjective generalized assignment problem, Advances in Engineering Software 38 (2007) 50–58.

D.F. Votaw, A. Orden, The personnel assignment problem, Symposium on Linear Inequalities and Programmng, SCOOP 10, US Air Force, 1952, pp. 155–163.
8. Fern, E. (2001). A Tabu search heuristic for the generalized assignment problem. *European Journal Of Operational Research*, *132*, 22-38.

Fisher, M.L., Jaikumar, R., 1981. A generalized assignment heuristic for vehicle routing. Networks 11, 109–124.

F.S. Hillier, M.M. Connors, Quadratic assignment problem algorithms and the location of indivisible facilities, Management Science 13 (1) (1966) 42–57.

G. Caron, P. Hansen, B. Jaumard, The assignment problem with seniority and job priority constraints, Operations Research 47 (3) (1999) 449–454.

G. Grygiel, Algebraic Rk-assignment problems, Control and Cybernetics 10 (3&4) (1981) 155–165.

G.T. Ross, R.M. Soland, A branch and bound algorithm for the generalized assignment problem, Mathematical Programming 8 (1975) 91–103.

Harper, P. R., de Senna, V., Vieira, I. T., & Shahani, A. K. (2005). A genetic algorithm for the project assignment problem. *Computers & Operations Research*, *32*(5), 1255-1265. doi:10.1016/j.cor.2003.11.003

H.R Kuhn. The Hungarian Method for Assignment Problem. Bryn Mawr College, 1955.

Jahanshahloo, G. R., & Afzalinejad, M. (2008). Goal programming in the context of the assignment problem and a computationally effective solution method. *Applied Mathematics and Computation*, *200*(1), 34-40. doi:10.1016/j.amc.2007.10.044

Jeet, V., & Kutanoglu, E. (2007). Lagrangian relaxation guided problem space search heuristics for generalized assignment problems. *European Journal of Operational Research*, *182*(3), 1039-1056. doi:10.1016/j.ejor.2006.09.060

J.S. Park, B.H. Lim, Y. Lee, A Lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem, Management Science 44 (1998) S271–S282.

Litvinchev, I., Mata, M., Rangel, S., & Saucedo, J. (2010). Lagrangian heuristic for a class of the generalized assignment problems. *Computers & Mathematics with Applications*, *60*(4), 1115-1123. Elsevier Ltd. doi:10.1016/j.camwa.2010.03.070

Lorena, L. A. N., & Narciso, M. G. (1996). Relaxation heuristics for a generalized assignment problem. *European Journal of Operational Research*, *91*(3), 600-610. doi:10.1016/0377-2217(95)00041-0

L.R. Ford Jr., D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, 1966 (Section II.6).

Michalewicz, Z. (1996). *Genetic Algorithms+Data Structures=Evolution Programs* (3rd ed.). Berlin, Germany: Springer.

Michalewicz, Z., Logan, T., & Swaminathan, S. (1994). Evolutionary operators for continuous convex parameter spaces. In A. V. Sebald & L. J. Fogel (Eds.), *Proceedings of 3rd Annual Conference on Evolutionary Programming* pp. 84–97). San Diego, CA: Singapore World Scientific.

M. Dell_Amico, S. Martello, The k-cardinality assignment problem, Discrete Applied Mathematics 76 (1–3) (1997) 103–121.

M. Laguna, J.P. Kelly, J.L. GonzEalez-Velarde, F. Glover, Tabu search for the multilevel generalized assignment problem, European J. Oper. Res. 82 (1995) 176–189.

M. Shigeno, Y. Saruwatari, T. Matsui, An algorithm for fractional assignment problems, Discrete Applied Mathematics 56 (2–3) (1995) 333–343.

Narciso, M. G., & Lorena, L. A. N. (1999). Lagrangean/surrogate relaxation for generalized assignment problems. *European Journal of Operational Research*, *114*(1), 165-177. doi:10.1016/S0377-2217(98)00038-1

Özbakir, L., Baykasoğlu, A., & Tapkan, P. (2010). Bees algorithm for generalized assignment problem. *Applied Mathematics and Computation*, *215*(11), 3782-3795. doi:10.1016/j.amc.2009.11.018

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, *176*(2), 774-793. doi:10.1016/j.ejor.2005.09.014

P. Kouvelis, G. Yu, Robust Discrete Optimization and Its Applications, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

P.T. Sokkalingam, Y.P. Aneja, Lexicographic bottleneck combinatorial problems, Operations Research Letters 23 (1) (1998) 27–33.

R.E. Burkard, Quadratic assignment problems, European Journal of Operational Research 15 (3) (1984) 283–289.

R.E. Burkard, F. Rendl, Lexicographic bottleneck problems, Operations Research Letters 10 (5) (1991) 303–308.

R.F. Subtil, E.G. Carrano, M.J.F. Souza, R.H.C. Takahashi, Using an enhanced integer NSGA-II for solving the multiobjective generalized assignment problem, in: 2010 IEEE World Congress on Computational Intelligence, pp. 1–7.

Romeijn, H. E., & Morales, D. R. (2000). A class of greedy algorithms for the generalized assignment problem. *Discrete Applied Mathematics*, *103*(1-3), 209-235. doi:10.1016/S0166-218X(99)00224-3

S. Geetha, K.P.K. Nair, A variation of the assignment problem, European Journal of Operational Research 68 (3) (1993) 422–426.

Shopova, E.G. & Vaklieva-Bancheva, N.G., 2006. BASIC—A genetic algorithm for engineering problems solution. *Computers & Chemical Engineering*, 30(8), p.1293-1309. Available at: http://linkinghub.elsevier.com/retrieve/pii/S009813540600055X [Accessed August 3, 2012].

S.K. Gupta, A.P. Punnen, Minimum deviation problems, Operations Research Letters 7 (4) (1988) 201–204.

S. Martello, W.R. Pulleyblank, P. Toth, D. de Werra, Balanced optimization problems, Operations Research Letters 3 (5) (1984) 275–278.

V. Balachandran, An integer generalized transportation model for optimal job assignment in computer networks, Operational Research 24 (1976) 742–759.

Woodcock, A. J., & Wilson, J. M. (2010). A hybrid tabu search/branch & bound approach to solving the generalized assignment problem. *European Journal of Operational Research*, *207*(2), 566-578. Elsevier B.V. doi:10.1016/j.ejor.2010.05.007.